

**UNIVERSIDAD AUTÓNOMA METROPOLITANA**  
**UNIDAD AZCAPOTZALCO**

**Maestría en Ciencias de la Computación**

**Área: procesamiento de señales y reconocimiento de patrones**

**Clasificación de imágenes usando redes  
neuronales convolucionales**

Idónea comunicación de resultados para obtener el grado de  
Maestro en Ciencias de la Computación

Presenta:

Lic. Fidel López Saca

Asesores:

Dr. Carlos Avilés Cruz

Dr. Andrés Ferreyra Ramírez

Sinodales: Dr. Sergio Gerardo de los Cobos Silva, Dra. Silvia Beatriz González Brambila,  
Dr. Carlos Avilés Cruz, Dr. Juan Villegas Cortez, M. C. Arturo Zúñiga López.

**30 de Mayo de 2019**



# Gracias

A la vida por darme la oportunidad y la fuerza para culminar este proyecto.

A mi mamá, hermano y abuela por la confianza, paciencia, apoyo y ánimo.

A todos mis amigos por su valiosa amistad a pesar de la distancia.

A las personas que han compartido una parte de su vida y experiencia.

## Agradecimientos

A través de estas líneas expreso mi profundo agradecimiento a mis asesores y grupo de trabajo, por su valioso tiempo, ayuda, paciencia, consejos, comentarios, sugerencias y críticas semanales durante el desarrollo de esta tesis.

A todos los profesores y miembros del Comité de la Maestría en Ciencias de la Computación por el tiempo invertido en la enseñanza.

A la Universidad Autónoma Metropolitana por darme la oportunidad de cumplir uno de mis sueños.

# Índice general

Índice de figuras	VII
Índice de tablas	XI
Lista de acrónimos	XII
Notación	XIV
Resumen	XVI
<b>1. Introducción</b>	<b>1</b>
1.1. Justificación . . . . .	3
1.2. Objetivo . . . . .	3
1.2.1. Objetivo general . . . . .	3
1.2.2. Objetivos específicos . . . . .	3
1.3. Antecedentes . . . . .	4
1.3.1. Redes neuronales convolucionales . . . . .	5
1.3.2. Aplicaciones con CNN . . . . .	9
<b>2. Fundamentos teóricos</b>	<b>15</b>
2.1. Identificación de imágenes . . . . .	15
2.2. Redes neuronales . . . . .	20
2.3. Redes neuronales convolucionales . . . . .	22
<b>3. Metodología</b>	<b>45</b>

3.1. Conjuntos de datos utilizados . . . . .	45
3.2. Acondicionamiento de los conjuntos de datos para entrenamiento y prueba . . . . .	46
3.3. Arquitectura propuesta . . . . .	49
3.4. Construcción de la red . . . . .	50
3.5. Entrenamiento de la red . . . . .	60
3.6. Prueba de la red . . . . .	62
<b>4. Resultados experimentales</b>	<b>65</b>
4.1. Primera prueba sin aumento de datos . . . . .	65
4.2. Segunda prueba con aumento de datos . . . . .	66
4.2.1. Rendimiento de redes por cada conjunto de imágenes . . . . .	68
4.3. Resultados del entrenamiento de la red ToniNet . . . . .	72
<b>5. Conclusiones y trabajo futuro</b>	<b>81</b>
<b>Bibliografía</b>	<b>83</b>
<b>A. Configuración y puesta en marcha de hardware y software</b>	<b>I</b>
A.1. Instalación de software . . . . .	I
A.2. Requerimientos técnicos . . . . .	II
<b>B. Toolbox para generar conjuntos de entrenamiento y prueba</b>	<b>V</b>
<b>C. Artículos</b>	<b>IX</b>
<b>D. Constancias</b>	<b>LVII</b>

# Índice de figuras

1.1. CNN con tres diferentes secciones de extracción de características. . .	2
1.2. Red neuronal artificial con las capas entrada, oculta y de salida. . . .	5
1.3. Arquitectura LeNet-5 de Yann LeCun [1]. . . . .	6
1.4. Arquitectura CNN, AlexNet [2]. . . . .	7
1.5. Comparación del error entre los ganadores en ILSVRC con el conjunto de datos ImageNet [3]. . . . .	9
2.1. Diagrama general de identificación de imágenes. . . . .	16
2.2. Ejemplo de clasificación de números MNIST [4]. . . . .	17
2.3. Clases que corresponden a distintas razas de perros. . . . .	18
2.4. Modelo de neurona o célula neuronal [5]. . . . .	21
2.5. Representación de neuronas artificiales. . . . .	22
2.6. Organización estructural de la IA, lugar donde se encuentra la CNN [6]. . . . .	22
2.7. Proceso que sigue la CNN. . . . .	24
2.8. Imagen en el espacio de color RGB representada por tres matrices, cada valor representa un pixel. . . . .	24
2.9. Ejemplo de un lote de seis imágenes que ingresan a la red. . . . .	25
2.10. Ejemplo de operación de convolución. . . . .	27
2.11. Ejemplo de operación max-pooling y average-pooling sobre la matriz $R$ . . . . .	29
2.12. Gráficas de las funciones de activación: (a) sigmoide, (b) tangente hiperbólica y (c) ReLU. . . . .	30

2.13. Resultados utilizando cinco capas, iniciando con una imagen de $256 \times 256$ y finalizando con imágenes de $8 \times 8$ . . . . .	31
2.14. Ejemplo de capas totalmente conectadas. . . . .	32
2.15. En (a) se muestra la red completa, en (b) se muestra el resultado del dropout. . . . .	33
2.16. Red utilizada como ejemplo para el Backpropagation. . . . .	35
2.17. Ejemplo de GPU, NVIDIA TITAN X [7]. . . . .	42
2.18. Rendimiento de GoogleNet con la comparación usando CPU-GPU [8].	43
3.1. Metodología de investigación para la construcción, entrenamiento y prueba de la red. . . . .	45
3.2. Imágenes Oliva & Torralba. . . . .	46
3.3. Imágenes Stanford Dogs. . . . .	46
3.4. Imágenes Caltech 256. . . . .	46
3.5. Ejemplos de imágenes de la clase costa, bosque y montaña. . . . .	47
3.6. Archivos que contienen imágenes, listos para ser ingresados a la red. .	48
3.7. Proceso para generar archivos tfrecord en TensorFlow. . . . .	48
3.8. Diseño de red neuronal convolucional. . . . .	50
3.9. Grafo de la arquitectura propuesta. . . . .	54
3.10. Gráfica con las diferentes pruebas de la red con el conjunto de imágenes de Oliva & Torralba. . . . .	55
3.11. Proceso para cargar archivos tfrecord en TensorFlow y generar lotes de imágenes. . . . .	60
3.12. Lectura de imágenes del dataset. . . . .	62
3.13. Proceso para realizar las pruebas. . . . .	63
4.1. Rendimiento de las redes en pruebas en top 1 con el conjunto de datos Oliva & Torralba. . . . .	68
4.2. Rendimiento de las redes en pruebas en top 5 con el conjunto de datos Oliva & Torralba. . . . .	69



4.3. Matriz de confusión con el conjunto de datos Oliva & Torralba, con las clases $C1, C2, \dots, C8$ que corresponden a <i>Opencountry, Coast, Forest, Highway, Inside_city, Mountain, Street</i> y <i>Tallbuilding</i> ; tiene un éxito en pruebas de 94.56 % con 765 imágenes correctas de 809. Resultados de la evaluación de la red entrenada ToniNet. . . . .	70
4.4. Prueba con Oliva & Torralba, la letra $R$ significa que es la clasificación real, la letra $P$ significa que es la clasificación dada por la red. . . . .	71
4.5. Rendimiento de las redes en pruebas en top 1 con el conjunto de imágenes Stanford Dogs. . . . .	72
4.6. Rendimiento de las redes en pruebas en top 5 con el conjunto de imágenes Stanford Dogs. . . . .	73
4.7. Rendimiento de las redes en pruebas en top 1 con el conjunto de imágenes Caltech 256. . . . .	74
4.8. Rendimiento de las redes en pruebas en top 5 con el conjunto de imágenes Caltech 256. . . . .	74
4.9. Disminución del error con diferentes conjuntos de imágenes usando la red ToniNet. . . . .	75
4.10. Rendimiento en entrenamiento en top 1 con la red ToniNet. . . . .	75
4.11. Rendimiento en entrenamiento en top 5 con la red ToniNet. . . . .	76
4.12. Entrenamiento con caltech, la letra $R$ significa que es la clasificación real, la letra $P$ significa que es la clasificación dada por la red. . . . .	77
4.13. Entrenamiento con Stanford Dogs, la letra $R$ significa que es la clasificación real, la letra $P$ significa que es la clasificación dada por la red. . . . .	78
4.14. Pruebas con Stanford Dogs, la letra $R$ significa que es la clasificación real, la letra $P$ significa que es la clasificación dada por la red. . . . .	79
A.1. Uso del software instalado para acceder a la GPU. . . . .	III
A.2. Servidor con 4 GPUs GTX 1080, utilizados para el entrenamiento de las redes. . . . .	IV

B.1. Inicio TFpyToolbox. . . . .	VI
B.2. Selección del conjunto de imágenes con TFpyToolbox. . . . .	VII
B.3. Generación de dataset. . . . .	VII
B.4. Escritura y lectura de los archivos tfrecord. . . . .	VII
D.1. Constancia COMIA 2018. . . . .	LVII
D.2. Constancia NEO 2017. . . . .	LVIII

# Índice de tablas

1.1. Capas de la CNN GoogleNet [9]. . . . .	13
3.1. Conjuntos de datos utilizados y sus características. . . . .	46
3.2. Conjuntos de imágenes separadas en entrenamiento y pruebas utilizando el 70 % de imágenes por clase para entrenamiento y el 30 % para pruebas. . . . .	49
3.3. Comparación con la red diseñada y redes usadas de diferentes kernel. . . . .	53
4.1. Resultados sin aumento de datos con 100 épocas y una GPU. . . . .	66
4.2. Resultados con aumento de datos y con 250 épocas, con cuatro GPUs. . . . .	67

## **Lista de acrónimos**

AI: Artificial Intelligence

ML: Machine Learning

DL: Deep Learning

CNN: Convolutional Neural Network

LSTM: Long Short-Term Memory

GPU: Graphics Processor Unit

RGB: Red, Green, Blue

ReLU: Rectified Linear Units

## Notación

$I$ : matriz que representa una imagen digital.

$X$ : matriz de datos de entrada.

$x_i$ : parámetros de entrada a la red.

$W$ : matriz de pesos sinápticos.

$w_{i,j}$ : pesos sinápticos.

$b_j$ : bias o valor del sesgo.

$K$ : matriz que representa el kernel en una convolución.

$R$ : matriz que representa la región en una imagen.

$M$ : matriz que representa la máscara en una región de la imagen.

$f$ : función de activación.

$C_H$ : convoluciones de forma horizontal.

$C_V$ : convoluciones de forma vertical.

$\rho$ : variable que representa el desplazamiento o paso al realizar la operación de convolución.

$\tau$ : variable que representa el relleno en una matriz.

$T$ : vector de clases.

$T_j$ : corresponde a un elemento  $j$  del vector de clases  $T$ .

$C$ : resultado de convolución.

$y^{(l)}(j)$ : representa un elemento de la capa totalmente conectada.

$s_j(x)$ : representa el resultado de la función de Softmax.

$m_t$ : estimación del primer momento con el sesgo corregido.

$v_t$ : estimación del segundo momento con el sesgo corregido.

$-\nabla J(\theta_t)$ : dirección negativa del gradiente o la derivada direccional de  $J(\theta_t)$ .

$\alpha$ : tasa de aprendizaje.

$\gamma$ : diferencia del gradiente anterior con el gradiente actual.

$\beta$ : lote de imágenes.

$\lambda$ : decaimiento de pesos.

## Resumen

La clasificación de imágenes de forma automatizada sigue siendo un problema abierto, es decir, no existe una solución al 100% fiable; al día de hoy se han trabajado diversas metodologías para resolver el problema. En los últimos años las redes neuronales convolucionales (CNN) han sido populares en el procesamiento de datos a gran escala y muchos trabajos han demostrado que son herramientas prometedoras en muchos campos, en especial, en la clasificación. En esta idónea comunicación de resultados se analiza, diseña y construye una nueva red neuronal convolucional para la clasificación de imágenes. Se seleccionan conjuntos de imágenes públicas y se separan para entrenamiento y prueba: con el primer conjunto se puede entrenar la red usando lotes de imágenes seleccionados de manera aleatoria; con el segundo conjunto de imágenes se valida el rendimiento.

Las CNN están formadas por dos secciones: extracción de características y clasificación. La ventaja de la red diseñada es que la sección de extracción de características utiliza tres kernel de convolución distintos por donde ingresa una imagen. Después se utilizan capas totalmente conectadas para la clasificación. Al realizar los experimentos se muestra que la red desarrollada aumenta el rendimiento al utilizar tres secciones de extracción de características pero incrementa el tiempo de entrenamiento.

La CNN se probó con tres conjuntos de imágenes y se hacen comparaciones contra las redes mundialmente conocidas como: AlexNet [2], GoogleNet [9] y ResNet 152 [10]. En pruebas la red diseñada proporcionó mejor precisión con los conjuntos de imágenes utilizados. Para esto se desarrolló un programa para generación de archivos *tfrecord*, utilizados principalmente para el entrenamiento de miles de imágenes, reduciendo el tiempo en el entrenamiento, también se utilizaron cuatro GPUs para el procesamiento de datos. Para poder utilizar las GPUs se ha tenido que implementar CUDA, se programó utilizando Python 2.7 y la librería TensorFlow.

**Palabras clave:** Reconocimiento de patrones, Redes neuronales convolucionales, Visión por computadora.





# Abstract

The image classification in an automated is an open problem, which means there is no reliable solution at 100%. During the last years, convolutional neural networks (CNN) have been popular in large-scale data processing and many studies have shown that they are promising tools in many fields, especially in image classification problem. In this results, a new CNN for the image classification is analyzed, designed and constructed. Public image datasets were selected, and processed for training and testing phases, such that the first dataset is used for CNN training, using image batches of randomly selected images; and the second image dataset the performance of the CNN is validated.

The CNNs are conformed by two sections: feature extraction and classification. The advantage of the designed network presented here is that the feature extraction section uses three different convolution kernels for image input. Then fully connected layers are used for classification purpose. By performing the experiments it is shown that the proposed network performance is increased by means of three feature extraction sections but training time in increased too.

CNN was tested with three image datasets and comparisons were made against worldwide known CNN such as: AlexNet, GoogleNet and ResNet 152. In the tests the designed network gave better precision with the used image datasets. For this, a program to generate files *tfrecord* was developed, used mainly for the training of thousands of images, reducing training time, four GPUs for data processing were also used.

In order to use the GPUs, CUDA had to be implemented, it was programmed using Python 2.7 framework and TensorFlow library.

Keywords: Pattern recognition, Convolutional Neural Networks, Computer vision.



# Capítulo 1

## Introducción

Conforme la capacidad de procesamiento de las computadoras y de otros dispositivos aumenta, se ha buscado que estos sean capaces de imitar algunas de las funciones que realizamos las personas. Uno de los sentidos que más interviene en nuestra vida diaria es el de la vista, en donde una gran cantidad de investigaciones y empresas se han sumado a desarrollar métodos para lograr que las computadoras puedan distinguir y clasificar diversas imágenes [11].

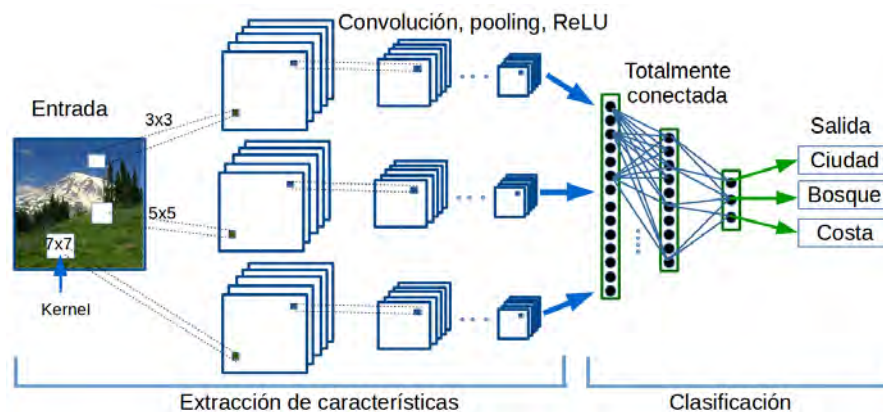
Una de las arquitecturas más populares utilizadas hoy en día es la *Red Neuronal Convolutiva* o Convolutional Neural Network (CNN). Las CNN que emulan la visión humana, tienen una alta precisión para la *clasificación de imágenes* [2], el *reconocimiento* [12] [13] y la *detección de objetos* [14], la *segmentación de escenas y el etiquetado* [15], tareas de la *visión artificial*. Se pueden implementar para el diagnóstico de enfermedades analizando las imágenes [16], en robótica, fabricación de automóviles, entre otras áreas. Para el diseño de estas redes existe información pero no a detalle, tampoco de sus implementaciones.

La convolución es el componente básico de una CNN y es por la que recibe su nombre. Es una operación entre una matriz de entrada y una matriz de filtros o kernel, el resultado es un mapa de características. La imagen es representada con una matriz (o matrices) y el kernel es la matriz de  $k \times k$  que se va modificando durante el proceso de aprendizaje [17]. Al realizar la operación de convolución en una imagen se obtiene el borde de figuras, el enfoque, entre otros, en el capítulo 2 se da más información.

El presente trabajo se centra en el desarrollo de una red neuronal convolutiva para la clasificación de imágenes. Se realiza un estudio del estado del arte de las

redes existentes para poder analizar y diseñar la propia red, después se construye y evalúa.

La principal característica de la red diseñada es que la imagen a clasificar ingresa a través de tres diferentes secciones de extracción de características cada una con cinco capas: la primera sección utiliza un kernel de tamaño  $3 \times 3$ ; la segunda sección utiliza un kernel de tamaño  $5 \times 5$ , en la siguiente capa el kernel se reduce a  $3 \times 3$  y se repite en las subsecuentes; en la tercera sección se inicia con un kernel de  $7 \times 7$ , en la siguiente capa se reduce a  $5 \times 5$ , después a  $3 \times 3$ . Así obtenemos información representativa de la imagen por cada kernel a lo largo de tres diferentes secciones. Finalmente, existen tres capas totalmente conectadas utilizadas para la clasificación como se muestra en la figura 1.1.



**Figura 1.1.** CNN con tres diferentes secciones de extracción de características.

Para realizar los entrenamientos se utilizaron cuatro unidades de procesamiento gráfico o GPUs NVIDIA GeForce GTX 1080, utilizando la librería TensorFlow [18] y se programó utilizando Python 2.7. Se necesitan imágenes clasificadas, separadas en entrenamiento y prueba.

Los resultados muestran que la red tiene un buen rendimiento en conjunto de imágenes como Oliva & Torralba [19], Stanford Dogs [20] y Caltech 256 [21]. Se realiza una comparación de la red propuesta con las redes existentes más competitivas, se compara contra AlexNet [2], GoogleNet [9] y ResNet [10].

Se ha podido demostrar que al realizar una red más profunda el tiempo de procesamiento aumenta, también necesita más imágenes para el aprendizaje. Las

redes con menor profundidad tienen un buen rendimiento con menos imágenes pero con menor precisión. Las redes se pueden implementar en diferentes dispositivos, como celulares, raspberry pi, jetson TX1, entre otros.

### 1.1. Justificación

La clasificación de imágenes es un problema abierto ya que aún no existe un método 100 % fiable que acredite el reconocimiento. Los métodos actuales de extracción de características se hacen de manera manual y dependen del experto, el problema es que si no se obtienen las características más representativas de la imagen es difícil obtener buenos resultados.

Las redes existentes normalmente se hacen profundas, esto quiere decir que se agregan múltiples capas a lo largo. La red que se diseñó crece tanto en profundidad como a lo ancho, extrayendo diferentes características con diferentes kernel en la operación de convolución. Después se combinan las características para ser ingresadas en las capas totalmente conectadas para realizar la clasificación.

La principal ventaja es que la imagen de entrada ingresa por diferentes capas, extrayendo diferentes y mejores características en el proceso, así se realiza una mejor clasificación, mostrando una nueva forma de realizar el diseño y construcción de redes. La red se puede ampliar a lo ancho utilizando otros métodos de extracción de características, no solo la convolución.

### 1.2. Objetivo

#### 1.2.1. Objetivo general

Analizar, diseñar, construir y evaluar una nueva red neuronal convolucional para la clasificación de imágenes.

#### 1.2.2. Objetivos específicos

- Analizar y diseñar una nueva arquitectura de una CNN.

- Construir y evaluar la nueva arquitectura propuesta.
- Realizar una comparación entre la arquitectura propuesta y otras redes, mostrando las ventajas y desventajas.
- Utilizar unidades de procesamiento gráfico (GPUs) para el procesamiento de datos.
- Desarrollar la arquitectura propuesta con la librería TensorFlow [18].

## Aportaciones

Las aportaciones de este trabajo son:

- Análisis de las redes neuronales convolucionales.
- Metodología para separar grandes cantidades de imágenes para entrenamiento y prueba.
- Una red neuronal convolucional para la clasificación de imágenes, extrayendo mejores características utilizando tres diferentes secciones.

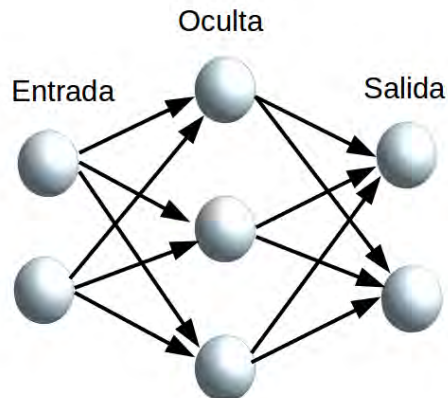
## Organización de la tesis

El trabajo de investigación está estructurado de la siguiente manera. En el capítulo 1 se revisan los antecedentes de las redes neuronales convolucionales, los tipos de redes y su implementación. En el capítulo 2 se introducen los fundamentos teóricos para la comprensión del trabajo. En el capítulo 3 se analiza la arquitectura de la red neuronal convolucional propuesta con la metodología para el desarrollo y la evaluación. En el capítulo 4 se muestran los resultados experimentales de los entrenamientos y pruebas. En el capítulo 5 se discute las conclusiones y trabajo futuro. Finalmente se muestran las referencias, glosarios y apéndices.

### 1.3. Antecedentes

En 1943 McCulloch y Pitts desarrollaron un modelo de la neurona biológica inspirada en la red neuronal del cerebro. Al conjunto de neuronas artificiales interconectadas

se le llama red neuronal artificial, en la figura 1.2 se muestra un ejemplo. La tarea principal es emular las habilidades para el reconocimiento de patrones del cerebro. Actualmente existen diferentes modelos de estas redes como: las redes neuronales recurrentes, redes neuronales convolucionales, entre otras.



**Figura 1.2.** Red neuronal artificial con las capas entrada, oculta y de salida.

En los últimos años las redes neuronales convolucionales han ganado fama, en especial en la competencia “ImageNet Large Scale Visual Recognition Challenge” (ILSVRC) [22] que se hace cada año desde el 2010, desde AlexNet[2] hasta ResNet[10] agregando nuevos algoritmos de aprendizaje y capas durante este proceso. Aunque Yann LeCun y su grupo de trabajo ya habían desarrollado las bases desde finales de los 90 del siglo pasado [1], con el avance del hardware han cobrado mayor fuerza, debido a que las GPUs pueden ser utilizadas para entrenar en horas una red, a comparación de hace 5 años que se hacía en días o semanas.

En las siguientes secciones se describen las redes que han dado mejores rendimientos para la clasificación y detección de objetos [22]. Después se describen en donde se han implementado. En el capítulo 2 se describe con más detalle la CNN y la diferencia con las redes neuronales artificiales.

### 1.3.1. Redes neuronales convolucionales

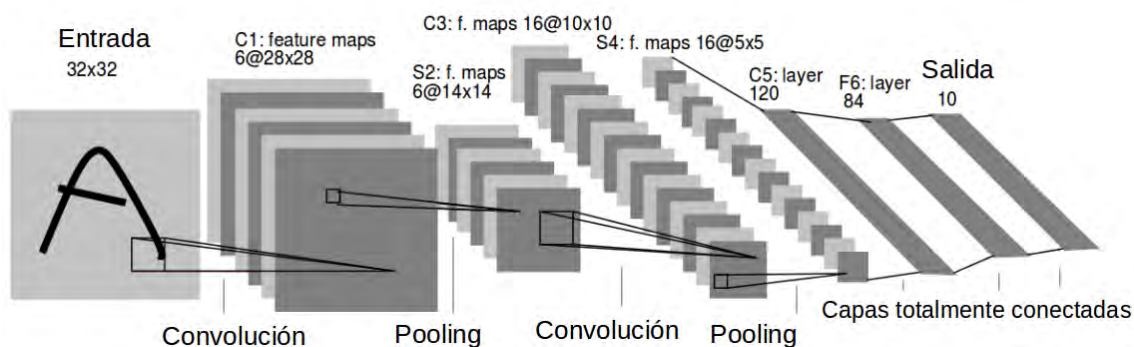
A continuación se presentan las investigaciones que sirven como base para esta tesis.

Las investigaciones se basan en trabajos para la creación de las propias redes o

utilizar las existentes para realizar transferencia de aprendizaje e implementarlas en diferentes dispositivos.

### LeNet-5

La arquitectura LeNet-5 [1] es una red neuronal convolucional usada para el reconocimiento de dígitos. La red tiene en la capa de entrada una imagen de  $32 \times 32$  píxeles. El total de capas son siete sin contar con la capa de entrada. En la figura 1.3 se muestra un diagrama de esta red.



**Figura 1.3.** Arquitectura LeNet-5 de Yann LeCun [1].

La primera capa es de convolución, con 6 kernel de  $5 \times 5$ , la salida de esta capa es de  $28 \times 28$ . En la capa dos reduce a  $14 \times 14$  extrayendo las mejores características con un kernel de  $2 \times 2$ , utiliza la función sigmoide como función de activación. En la capa 3 utiliza 16 kernel de  $5 \times 5$  reduciendo la imagen a  $10 \times 10$ . En la capa cuatro extrae las características con 16 kernel de  $2 \times 2$  reduciendo a  $5 \times 5$  la imagen. Para la clasificación utiliza una capa totalmente conectada de 120 neuronas o nodos, después 84 y 10 ya que son los dígitos a clasificar.

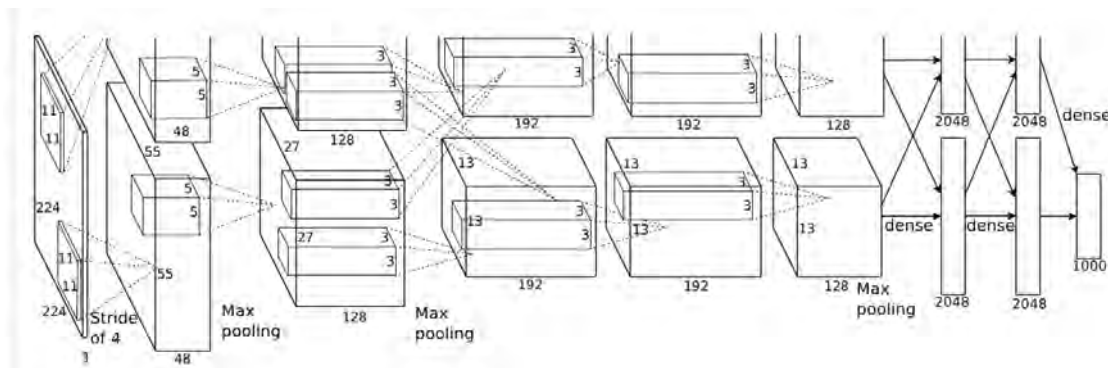
### AlexNet

En el concurso ILSVRC del año 2012 se hizo famosa la arquitectura de red neuronal convolucional llamada AlexNet [2], dicha arquitectura se muestra en la figura 1.4. La red clasificó mil diferentes clases, entrenada con el conjunto de imágenes de ImageNet [23] que contiene más de un millón de imágenes de alta resolución, obtuvo un error en top-5 de 15.3%. Para describir la precisión de las redes en la tarea de



clasificación utilizamos los términos **top-1** y **top-5**. El **top-5** es la cantidad de veces que la etiqueta correcta está dentro de las 5 clases principales predichas por la red. El **top-1** es la cantidad de veces que la etiqueta correcta tiene la probabilidad más alta predicha por la red.

AlexNet consta de cinco capas convolucionales, algunas de las cuales son seguidas de capas de max-pooling utilizada para reducir el tamaño de la imagen, también incluye tres capas totalmente conectadas. Para realizar el entrenamiento de forma rápida usaron dos GPUs. El sobre-entrenamiento se da cuando la red se ajustada a parámetros muy específicos del entrenamiento, obteniendo alto rendimiento en entrenamiento pero bajo en pruebas. Para reducirlo, en las capas completamente conectadas se utilizó el método *dropout* para simular diferentes redes, deshabilitando de forma aleatoria un porcentaje de los nodos. Con respecto a LeNet-5 esta red incrementó la cantidad de capas y filtros mejorando la precisión en la clasificación de imágenes.



**Figura 1.4.** Arquitectura CNN, AlexNet [2].

## GoogleNet

En el año 2014 Christian Szegedy y sus colaboradores propusieron una arquitectura de red neuronal convolucional participando en el concurso de ILSVRC, en el diseño aumentaron la profundidad y el ancho de la red. Tiene el nombre de GoogleNet [9], una red de 22 capas de profundidad que incluyen bloques de subcapas, en total son alrededor de cien capas cuya calidad se evalúa en el contexto de la clasificación y la detección. La imagen de entrada tiene un tamaño de  $224 \times 224 \times 3$  píxeles, en

la primera capa tiene 64 kernel de  $7 \times 7$  con un paso de 2, dando como resultado  $64 \times 112 \times 112$  características. Luego aplica un max-pooling con un kernel de  $3 \times 3$  y un paso de 2. Después se aplica una convolución con un kernel de  $3 \times 3$  repetido 192 veces y posteriormente vuelve aplicarse un max-pooling de  $3 \times 3$ . También contiene dos capas de inicio (3a) y (3b), estos contienen subcapas. En la tabla 1.1 se muestra las características de las capas de esta arquitectura.

La red de GoogleNet incrementó la cantidad de capas con respecto a AlexNet haciéndola más robusta pero mejorando la precisión en las pruebas.

### ResNet

ResNet [10] es una red de 152 capas de profundidad, tiene menos complejidad que GoogleNet pero mejor precisión con un error de **top-5** de 3.57% en el conjunto de pruebas de ImageNet, tiene diseños con 50 y 101 capas, dando mejor resultado la de 152. También tiene bloques internos con filtros entre 64 y 512.

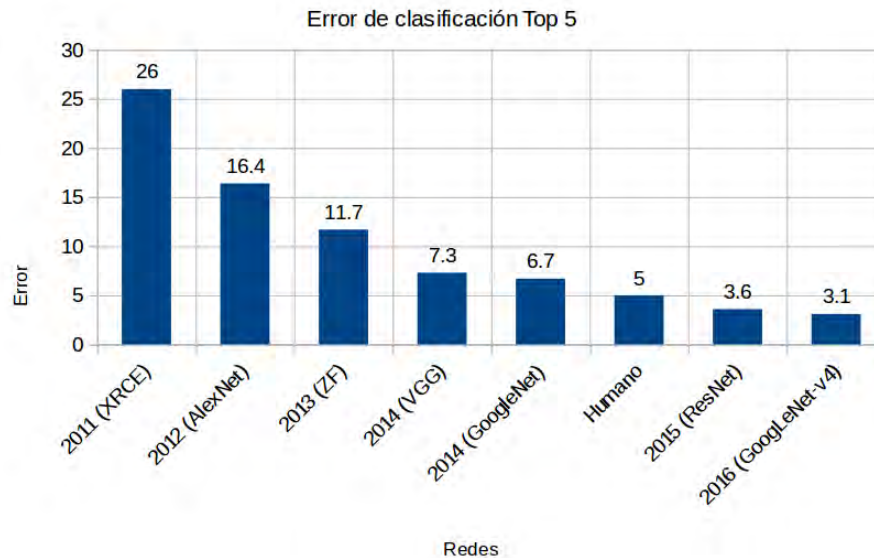
Para entrenar con el conjunto de imágenes de ImageNet se hace un preprocesamiento de datos antes del ingreso a la red, de tal manera que la imagen de entrada sea de  $224 \times 244$  pixeles. Utiliza la normalización por lotes y la tasa de aprendizaje comienza desde 0.1 y disminuye en factor de 10 a medida que el error se mantiene, utilizado para tener cambios más específicos del aprendizaje. El decaimiento de pesos o factor de regularización fue de 0.0001, no utilizaron dropout.

ResNet al tener más capas que AlexNet y GoogleNet incrementa el tiempo de entrenamiento, esta es una desventaja si se tiene gran cantidad de imágenes pero da mejor precisión.

### Comparación de redes

Las capas han aumentado desde LeNet-5 hasta ResNet-152, mejorando el rendimiento en los conjuntos de imágenes utilizados. En la figura 1.5 se muestra la disminución de error con respecto al concurso de ILSVRC a través de los años. Las redes con más profundidad han podido entrenarse con datasets como ImageNet apoyándose de poderosas GPUs. Pareciera que a mayor profundidad da

mejores resultados, pero también depende del conjunto de imágenes, los parámetros utilizados, los algoritmos de aprendizaje, etc.



**Figura 1.5.** Comparación del error entre los ganadores en ILSVRC con el conjunto de datos ImageNet [3].

### 1.3.2. Aplicaciones con CNN

Las CNN pueden ser utilizadas para la clasificación de imágenes, detección de objetos, segmentación, búsqueda de imágenes, combinación de características entre imágenes, etc. Se pueden implementar para el análisis de imágenes vía satélite [24], en medicina para detección de cáncer en la piel [25], [26], para la detección de tráfico [27], para el manejo de autos [28], entre otras implementaciones. A continuación se presentan unas aplicaciones relevantes.

#### Detección de gestos con las manos

Los pacientes con accidente cerebro-vascular, tienen con frecuencia un problema de un lado del cuerpo (hemiplejia), y tienen que realizar entrenamiento para su rehabilitación. En el artículo [29] describen el desarrollo de una plataforma de entrenamiento de locomoción, basado en el reconocimiento de gestos con las manos. El entrenador usa el sistema, con el objetivo de controlar la plataforma para mejorar

la rehabilitación. Implementan la arquitectura AlexNet modificando la última capa de la red para poder entrenarla al número de clases utilizadas. Describen el uso de 600 imágenes divididas en cinco clases las cuales son: avanzar, retroceder, izquierda, derecha y detener. Cada clase contiene 120 imágenes, al ingresarlas a la red las redimensionan para tener una imagen de  $227 \times 227 \times 3$ . Las clases son utilizadas para poder controlar la plataforma de entrenamiento, con una precisión de 99.98 % para el reconocimiento de los gestos.

### **Detección de incendios**

Detectar a tiempo el fuego puede evitar la pérdida de objetos o de vidas. En el trabajo [30] proponen un modelo para detección de incendios usando cámaras de videovigilancia, inspirado en GoogleNet. Realizaron el entrenamiento con 14 vídeos donde hay incendios y 17 vídeos donde no hay. También hicieron pruebas con diferentes imágenes. Utilizaron una GPU NVidia GeForce GTX TITAN X con 12 GB de memoria RAM, con un CPU Intel Core i5 con 64 GB de RAM y sistema operativo Ubuntu. Para entrenar la red utilizaron una tasa de aprendizaje de 0.001, obtuvieron una precisión de 94.43 % en pruebas.

### **Clasificación de personas entre edad y género**

Otra aplicación con GoogleNet es un sistema de clasificación de edad y género [31]. El sistema tiene una velocidad de 8 cuadros o frames por segundo con una precisión de 98 % en un escenario real. Utilizaron una tarjeta NVIDIA Jetson TX1 para la implementación. Para el entrenamiento utilizaron un conjunto de imágenes de 26,580 con 2,284 personas. Están divididos en ocho grupos de intervalos: 0-2, 4-6, 8-13, 15-20, 25-32, 38-43, 48-53, 60+ años de edad.

### **Detección de peatones**

Para los vehículos autónomos es importante la detección de peatones para evitar algún accidente. En el artículo [32] proponen un método para predecir la detección de peatones y el nivel de peligro (alto, bajo, sin peligro) con ResNet. La efectividad

que se logró fue de 68 % de precisión con un tiempo de procesamiento de 58.6 frames por segundo. Para el entrenamiento usaron 1,316 videos. La tasa de aprendizaje fue inicializada con 0.1 y cada 30 épocas disminuye en un factor de 10. Utilizaron el gradiente descendente estocástico con momento de 0.9. La disminución de peso fue de 0.0001. Para realizar el entrenamiento y las pruebas utilizaron un GPU NVIDIA GeForce GTX 1080.

### **Aplicaciones de CNN en dispositivos móviles**

En aplicaciones de reconocimiento en tiempo real, las redes pequeñas (con pocas capas) han sido implementadas en dispositivos móviles con pocos recursos de memoria y procesamiento, por ejemplo: MobileNet [33] entrenada con ImageNet [23] para la clasificación de 1,000 objetos diferentes. Es factible entrenar la red con servidores que tenga grandes recursos en procesadores, memoria, GPUs, entre otros, después se puede implementar la red entrenada en dispositivos con pocos recursos. MobiliNet se ha implementado en smartphones con sistema operativo Android para detectar la retinopatía diabética, entrenada y probada con más de 16,000 imágenes preprocesadas para la detección de este problema [34]. Sin embargo, redes muy grandes (con muchas capas) no pueden ser implementadas en dispositivos móviles, ni en sistemas embebidos de bajo costo tales como: Arduino, Beagle y Raspberry, ya que sus procesadores de propósito general no han sido optimizados para la implementación de estas redes.

### **Otras aplicaciones con CNN**

La CNN se puede utilizar para la *segmentación* de imágenes, esta detecta el borde de los objetos y los rellena de un color característico [35], puede ser utilizado en los autos autónomos para realizar una conducción automática, para sistemas de realidad virtual, entre otros [36].

Otra aplicación es el *auto-encoder* [37], puede ser utilizado para realizar búsquedas de imágenes [38], [39], extrayendo las mejores características de un conjunto de imágenes de entrenamiento y transformándolas en un vector binario,

utilizado para comparar con el vector de la imagen de búsqueda. Está formada por el *encoder* o codificador utilizado para transformar el vector las características que representa la imagen y el *decoder* o decodificador para la reconstrucción de las características. Del vector de bits depende la precisión de búsqueda, puede ser muy pequeña, por ejemplo de doce bits o más grande para que la búsqueda dé mejores resultados, sin olvidar que la CNN es la que extrae las mejores características de un conjunto de imágenes.

También las CNNs pueden crear nuevas imágenes, combinando las características entre distintas imágenes. Se puede entrenar una red para la transferencia de información de estilo, color, tono y textura de una imagen a otra [40], [41], [42].

type	patch size/ stride	output size	depth	# $1 \times 1$	# $3 \times 3$ reduce	# $3 \times 3$	# $5 \times 5$ reduce	# $5 \times 5$	pool proj	params	ops
convolution	$7 \times 7/2$	$11 \times 112 \times 64$	1							2.7K	34M
max pool	$3 \times 3/2$	$56 \times 56 \times 64$	0								
convolution	$3 \times 3/1$	$56 \times 56 \times 192$	2		64	192				112K	360M
max pool	$3 \times 3/2$	$28 \times 28 \times 192$	0								
inception (3a)		$28 \times 28 \times 256$	2	64	96	128	16	32	32	159K	128M
inception (3b)		$28 \times 28 \times 480$	2	128	128	192	32	96	64	380K	304M
max pool	$3 \times 3/2$	$14 \times 14 \times 480$	0								
inception (4a)		$14 \times 14 \times 512$	2	192	96	208	16	48	64	364K	73M
inception (4b)		$14 \times 14 \times 512$	2	160	112	224	24	64	64	437K	88M
inception (4c)		$14 \times 14 \times 512$	2	128	128	256	24	64	64	463K	100M
inception (4d)		$14 \times 14 \times 528$	2	112	144	288	32	64	64	580K	119M
inception (4e)		$14 \times 14 \times 832$	2	256	160	320	32	128	128	840K	170M
max pool	$3 \times 3/2$	$7 \times 7 \times 832$	0								
inception (5a)		$7 \times 7 \times 832$	2	256	160	320	32	128	128	1072K	54M
inception (5b)		$7 \times 7 \times 1024$	2	384	192	384	48	128	128	1388K	71M
avg pool	$7 \times 7/1$	$1 \times 1 \times 1024$	0								
dropout (40%)		$1 \times 1 \times 1024$	0								
linear		$1 \times 1 \times 1000$	1							1000K	1M
softmax		$1 \times 1 \times 1000$	0								

Tabla 1.1. Capas de la CNN GoogleNet [9].





# Capítulo 2

## Fundamentos teóricos

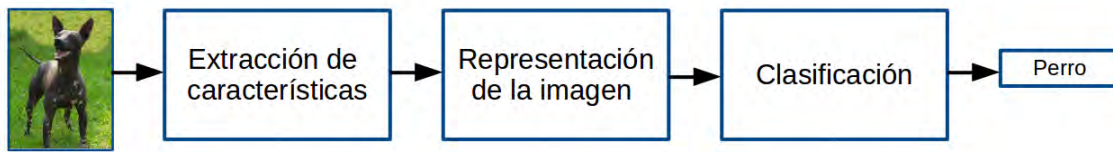
En este capítulo se describen los fundamentos teóricos para la comprensión del trabajo. En un inicio se aborda el tema de clasificación de imágenes, después los temas relacionados con las redes neuronales; primero la red neuronal biológica, luego la red neuronal artificial básica, para finalmente, abordar la red neuronal convolucional.

### 2.1. Identificación de imágenes

Los seres vivos cuentan con el instinto básico de identificar y clasificar el entorno acorde a sus sentidos. En el caso de la visión artificial, especialmente el de imágenes, en general la *clasificación* tiene por objetivo asignar una categoría a un conjunto finito de clases o etiquetas, es posible reconocer las características dominantes de la imagen usando métodos estadísticos. La clasificación se refiere al procedimiento en el que a un nuevo elemento se le asigna una clase predefinida en función de sus características observadas [43]. En otras palabras, el objetivo de la clasificación de imágenes es asignar a una imagen la etiqueta que le corresponde.

La clasificación de imágenes se puede realizar por diferentes métodos como: máquinas de soporte vectorial, árboles de decisión, redes neuronales, entre otros; para más detalle véase [44]. A continuación se describe cada módulo del diagrama general de la clasificación típica de imágenes [45], como se muestra en la figura 2.1.

- **Extracción de características:** las características son los elementos que describen un objeto, como el tamaño, la forma, el color, etc. La extracción de características es un factor importante para lograr un buen rendimiento.



**Figura 2.1.** Diagrama general de identificación de imágenes.

Normalmente las características más representativas se encuentran localizadas en zonas con cambios de contraste o en los contornos de los objetos. El extractor de características contiene la mayor parte del conocimiento previo y es bastante específico para cada tarea de clasificación. Seleccionar una técnica adecuada debe hacerse con cuidado. Algunos métodos de extracción de características se describen en [46].

- **Representación de la imagen:** desde el punto de vista estadístico, se realiza una representación única de toda la imagen en forma de arreglo numérico. Esta representación se obtiene sumando o combinando cada uno de los vectores numéricos que describen a cada uno de los puntos de interés.
- **Clasificación:** se realiza la clasificación de la imagen con base en su representación única. Es en esta etapa donde el sistema de clasificación aprende del conjunto de entrenamiento a distinguir las imágenes de las diferentes categorías a partir de la representación visual de cada imagen.

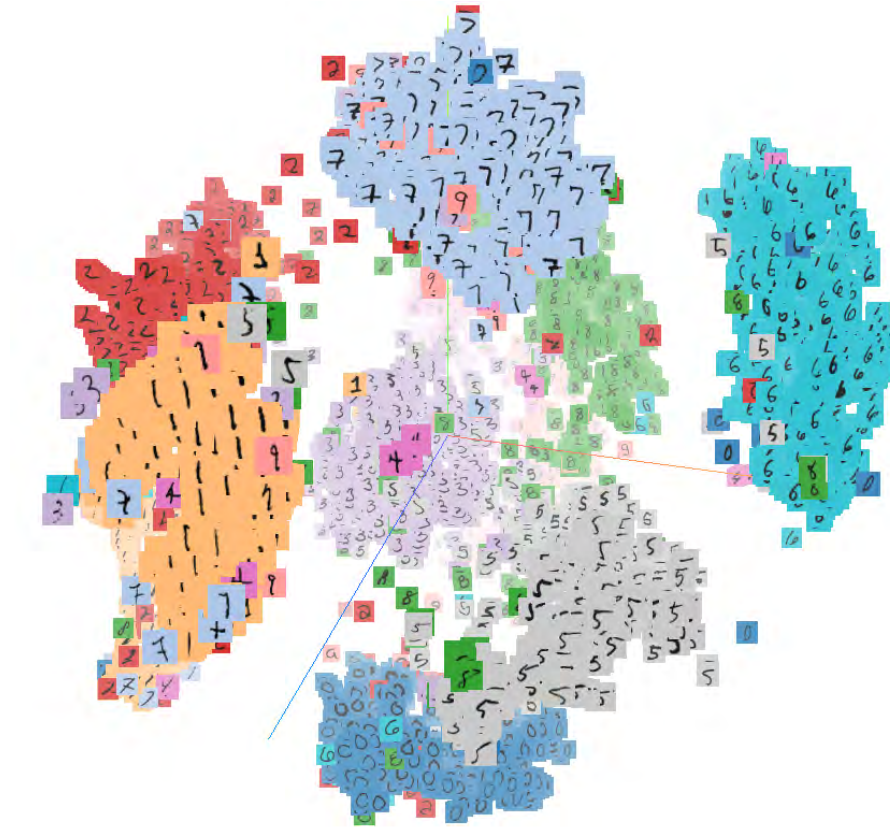
En la figura 2.2 [47] se puede ver de forma gráfica la separación de los datos para la clasificación.

Los conjuntos de imágenes que utilizamos en este trabajo para que la red aprenda están clasificados por humanos, separadas en diferentes carpetas a las que llamaremos **clases**, e. g. la raza de perro pekinés y papillón de la figura 2.3.

### Tipos de clasificadores

Existen diferentes tipos de clasificadores [48] como los que se describen a continuación.

- **Clasificadores paramétricos:** este tipo de clasificadores se basan en funciones de densidad de probabilidad, por ejemplo una distribución normal.



**Figura 2.2.** Ejemplo de clasificación de números MNIST [4].

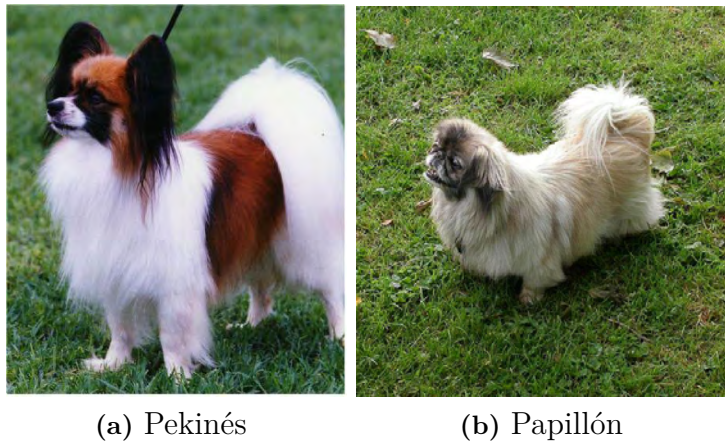
Al realizar el entrenamiento se generan los parámetros como las medias y las covarianzas. Cuando es difícil clasificar a menudo generan resultados con ruido. Ejemplo de clasificadores: clasificador gaussiano, máxima probabilidad, análisis discriminante lineal [48].

- **Clasificadores no paramétricos:** los clasificadores no paramétricos evitan emplear parámetros estadísticos para realizar la separación de clases.

Ejemplo de estos clasificadores: Knn, red neuronal artificial, árbol de decisión, máquina de soporte vectorial [48].

## Métodos de aprendizaje

Aprender es adquirir el conocimiento de algo, ya sea a través del estudio o experiencia, entonces aprendizaje es el tiempo durante el que se aprende algo [49]. En el libro de inteligencia artificial [50] mencionan que el aprendizaje se produce



**Figura 2.3.** Clases que corresponden a distintas razas de perros.

como resultado de la interacción entre un agente y el mundo y de la observación por el agente de sus propios procesos de toma de decisiones.

Existen diferentes tipos de métodos de aprendizaje como se describen a continuación:

- **Aprendizaje supervisado:** La principal característica es que las entradas son validadas con respecto a las salidas esperadas o correspondiente a ellas. El entrenamiento es utilizado para generar el conocimiento y los elementos de prueba para validarlo donde los elementos pertenecen al mismo conjunto [48]. Los resultados generados a partir de las muestras de entrenamiento son usados para realizar la evaluación de las pruebas con datos parecidos a los del entrenamiento.
- **Aprendizaje no supervisado:** este tipo de aprendizaje solo tiene datos de entrada, no se tiene información sobre las clasificaciones o salidas esperadas [48]. Los algoritmos tienen que ser capaces de reconocer características para las nuevas entradas. La información debe de ser validada por el experto en el tema para poder evaluar su rendimiento.

### Algoritmos de aprendizaje

- **Máquinas de soporte vectorial:** las máquinas de soporte vectorial o Support Vector Machines (SVM) son algoritmos de aprendizaje supervisado,

son usados para clasificación y regresión [51]. Usan condiciones lineales para separar las clases entre sí.

- **Árboles de decisiones:** los árboles de decisión crean particiones de manera jerárquica. De manera general intenta dividir los datos de entrenamiento para maximizar las diferencias entre clases usando diferentes nodos [52].
- **Algoritmos genéticos:** son algoritmos de búsqueda y aprendizaje, utilizan el cruzamiento y la mutación para generar nuevos elementos. Inspirados en el principio de evolución genética [52]. También se les llaman algoritmos evolutivos y su eficiencia depende de los parámetros establecidos como las poblaciones sucesivas (generaciones). La evolución está dada por una selección y operadores genéticos como la mutación y cruza que producen una nueva generación de individuos [53].
- **K-means:** es un algoritmo de agrupamiento. Primero selecciona k objetos al azar, donde cada objeto representa un clúster, después trata de encontrar los elementos más cercanos haciendo cálculos de distancia desde el centro del cluster y son asignados a los grupos más cercanos [54].
- **Redes neuronales artificiales:** son modelos matemáticos que están inspirados en las neuronas biológicas e intenta simular el cerebro humano [52].

## Aprendizaje profundo

El *aprendizaje profundo* o Deep Learning (DL) permite aprender características de imágenes que se van adaptando al conjunto de datos a medida que se va entrenando. DL es un subcampo del *aprendizaje automático*, permite que los modelos computacionales compuestos de múltiples capas de procesamiento aprendan representaciones de datos con múltiples niveles de abstracción [55]. Actualmente es una de las técnicas de aprendizaje artificial más utilizada, ya que emplea un conjunto de algoritmos que trabajan en múltiples capas para aprender representaciones a partir de datos. En una imagen los datos pueden ser un vector de píxeles; al analizar los datos de las imágenes se tendrán que analizar todos los parámetros pero, sólo algunos son los que hacen más fácil aprender sobre la tarea que nos interesa, e. g.,

en una recámara identificar una cama como objeto.

Al ver las capas conectadas las podemos relacionar como una red de neuronas artificiales profundas [56]. El artículo “Deep learning in neural networks: An overview” [57] describe parte de su historia.

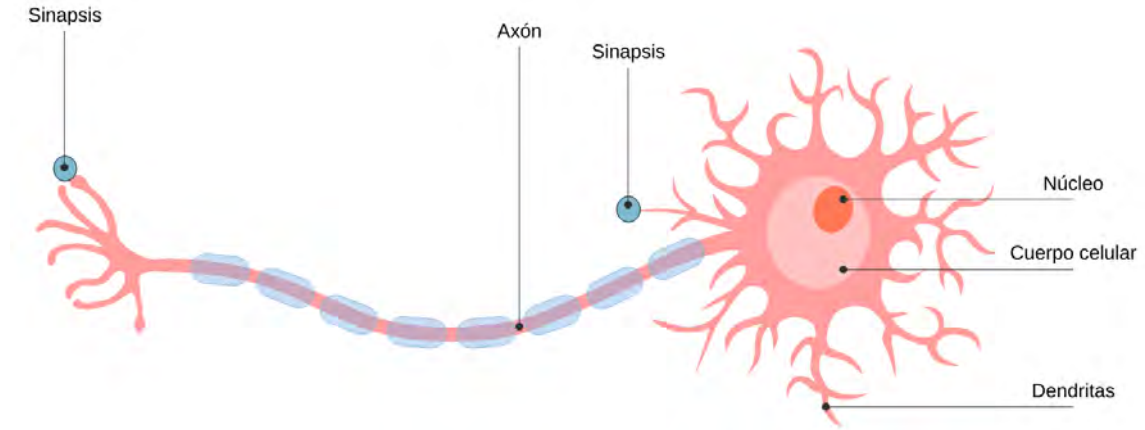
### 2.2. Redes neuronales

En 1943 McCulloch y Pitts [58] realizaron un modelo del comportamiento de la neurona biológica, la cual es la base para construir una red de neuronas artificiales. A finales de 1950 Frank Rosenblatt [59] y otros investigadores desarrollaron una clase de redes neuronales llamadas *perceptrones* basadas en el modelo de McCulloch-Pitts. En los perceptrones se agregó una regla de aprendizaje para el entrenamiento de redes, esto para resolver problemas de reconocimiento de patrones. En 1960 Bernard Widrow y Marcial Hoff desarrollaron un modelo de nombre Adaline (*Adaptive Linear Elements*). Después fue diseñado el perceptrón multicapa utilizando el backpropagation publicado por Rumelhart en 1986 utilizado para el aprendizaje.

Las redes neuronales pueden ser modeladas de manera biológica y también de manera artificial como se describe a continuación.

#### Redes neuronales biológicas

Las partes utilizadas para modelar la red neuronal biológica [60] son: las dendritas, la sinapsis, el cuerpo celular y el axón. La información llega a través de las dendritas, con las conexiones llamadas sinapsis que sirven como unión entre neuronas. Después, la información pasa por el cuerpo celular para ser procesada. Este cuerpo celular da resultados a la salida utilizando unas fibras largas llamadas axón que sirven para enviar información a otra neurona. En la figura 2.4 se puede ver una neurona biológica perteneciente al sistema nervioso central. Al conjunto de neuronas interconectadas se le llama red neuronal.



**Figura 2.4.** Modelo de neurona o célula neuronal [5].

## Redes neuronales artificiales

En los últimos años el enfoque de la red neuronal artificial ha sido ampliamente adoptado; el hardware ha ayudado especialmente en disminuir los tiempos de entrenamiento. La red neuronal tiene ventajas como: no es paramétrica, es adaptable a diferentes conjuntos de datos, capacidad de generalización para diferentes tipos de imágenes, entre otros aspectos. El perceptrón multicapa es el modelo más popular de red neuronal en la clasificación de imágenes basada en la red neuronal biológica. Los parámetros de entrada son las  $x_i$ , los pesos son los  $w_{i,j}$  y los umbrales  $b_j$  utilizados para realizar la sumatoria representada en la ecuación 2.1, y después pasa por una función de activación para obtener las salidas  $y_j$ , donde  $j$  es un elemento de  $r$  neuronas. En la figura 2.5 se puede ver la representación de la red neuronal artificial modelando las partes de una red neuronal biológica.

$$\sum_{i=1}^q w_{i,j} x_i + b_j \quad (2.1)$$

Donde:

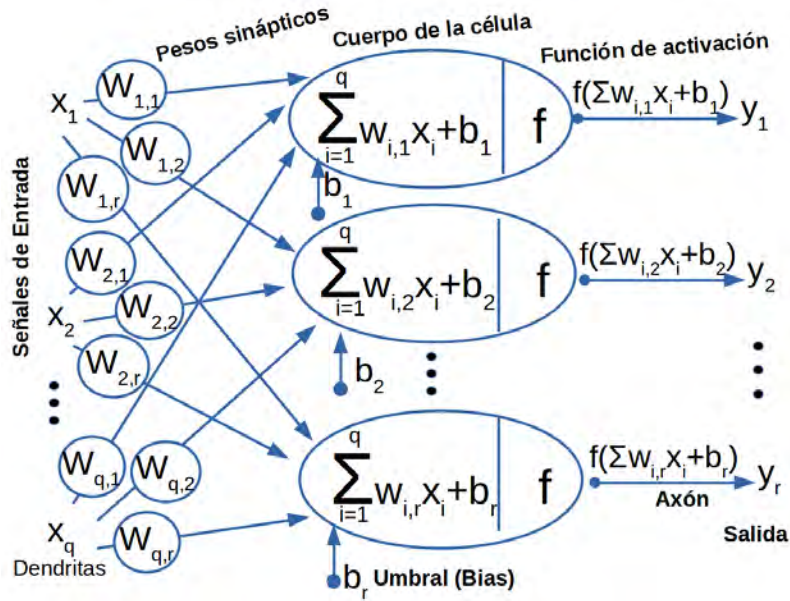
$x_i$ : son los parámetros de entrada,

$w_{i,j}$ : son los pesos sinápticos,

$b_j$ : es el umbral,

$q$ : cantidad de parámetros de entrada.

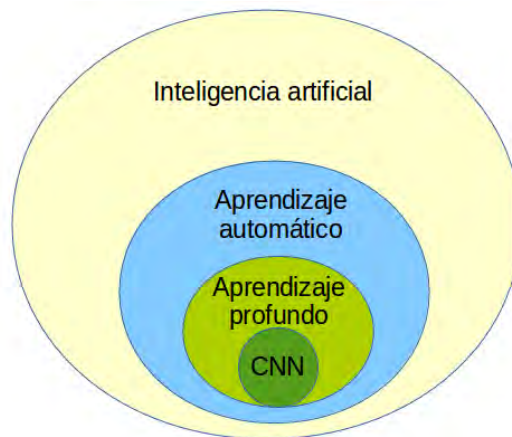




**Figura 2.5.** Representación de neuronas artificiales.

## 2.3. Redes neuronales convolucionales

Entre las distintas redes se encuentra una que nos interesa, la *red neuronal convolucional* (RNC) o Convolutional Neural Network (CNN). Como se muestra en la figura 2.6, estas redes se encuentran en el área de DL por estar diseñadas en forma de redes neuronales profundas.



**Figura 2.6.** Organización estructural de la IA, lugar donde se encuentra la CNN [6].

Las CNN se inspiraron en la estructura del sistema visual. Al día de hoy

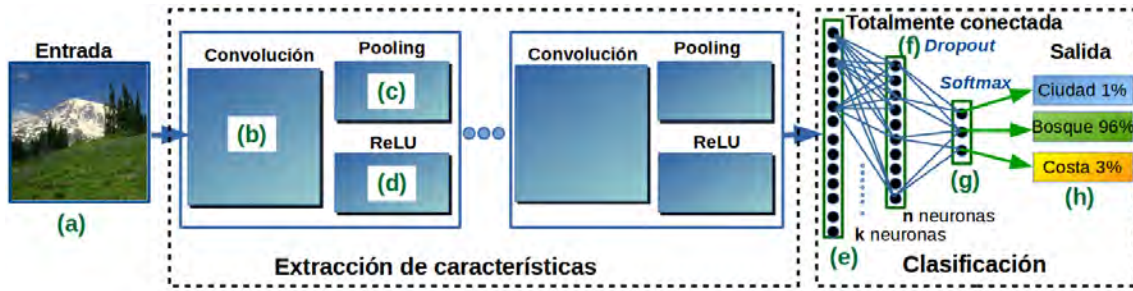


se encuentran entre los sistemas con mejores resultados para el reconocimiento de patrones [61]; son utilizados para el procesamiento de imágenes, vídeo, voz y audio. Estas redes han demostrado ser una herramienta posible de implementarse en distintas áreas; en medicina para la segmentación de imágenes de ultrasonido musculo-esquelético [62] o para la detección de sangrado gastrointestinal [63]. También se puede implementar en la industria automotriz [64], entre otras áreas.

La CNN es una red neuronal profunda, tiene entre sus capas a la operación de convolución que realiza una operación entre las matrices de la imagen y una matriz llamada kernel, utilizada para obtener las características como son borde, enfoque, entre otras. Las CNNs son parte del aprendizaje supervisado pero, existen otras redes como las *redes neuronales recurrentes* (empleadas para el texto y el habla [55], entre otros [65]) utilizan el aprendizaje no supervisado. Se espera que las redes neuronales recurrentes sean mucho más importantes a largo plazo ya que el aprendizaje humano se hace de ese modo de acuerdo con LeCun y Bengio [55].

## Elementos de una CNN

En general la CNN es una estructura jerárquica de red neuronal artificial compuesta por los siguientes elementos: entrada, convolución, pooling, ReLU, una capa totalmente conectada y al final la capa de salida. Las capas de convolución y reducción pueden repetirse varias veces dependiendo del modelo y son utilizadas para obtener las características más representativas de la imagen. La capa totalmente conectada, puede ser una o varias, unifica los resultados para realizar la clasificación. Al final en la capa de salida se muestra que tan probable es que la imagen de entrada, por ejemplo, sea una ciudad, un bosque o una costa. En la figura 2.7 se muestra el proceso que sigue la CNN donde se encuentra la imagen de entrada (a), después pasa por la capa de convolución (b), luego se reduce con pooling (c) y pasa por una función de activación como ReLU (d). Al final de la figura se encuentra la capa totalmente conectada (e) con las funciones de dropout (f) y softmax (g), dando una salida (h) para obtener la clasificación.



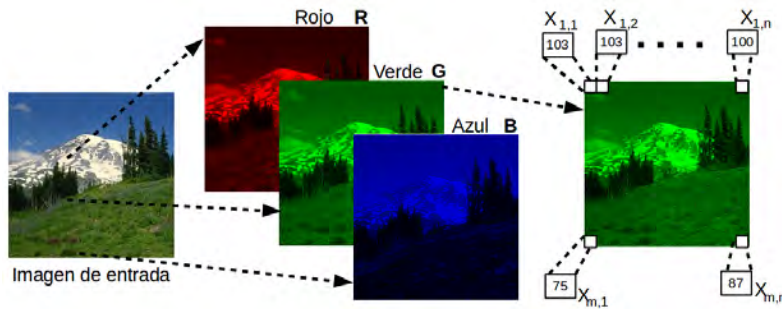
**Figura 2.7.** Proceso que sigue la CNN.

A continuación se describe cada paso del proceso.

## Extracción de características

### (a) Capa entrada de la imagen

Las imágenes se pueden representar como un vector bidimensional (una matriz) de  $M \times N$  ( $M$  representa el ancho y  $N$  el alto), donde cada valor de la matriz representa un píxel. Si se trabaja en el espacio RGB «Red, Green, Blue», serían 3 matrices de  $M \times N$ , teniendo una matriz por canal. En la figura 2.8 se muestra un ejemplo de una imagen representada por sus matrices componentes.



**Figura 2.8.** Imagen en el espacio de color RGB representada por tres matrices, cada valor representa un píxel.

El proceso de identificación de una imagen con CNN inicia con el ingreso de la imagen en formato RGB (puede ser otro espacio de color, depende del modelo propuesto), en éste espacio de color se generan mejores resultados [66]. Dado que cada imagen tiene tamaño diferente se normalizan a un tamaño de  $N \times N$ , esto se hace para que sea fácil tratar los datos en la red. Se puede considerar como una capa

de preprocesamiento donde las imágenes de entrada pueden ser redimensionadas, giradas o submuestreadas.

En la figura 2.9 se muestra un ejemplo de la entrada a la red con lote de seis imágenes con las descripciones de la clase a la que pertenece. Las imágenes fueron tomadas del conjunto de datos de Oliva & Torralba [19].



**Figura 2.9.** Ejemplo de un lote de seis imágenes que ingresan a la red.

### (b) Convolución

Con los datos de la imagen normalizada y representados en matrices se puede aplicar la operación de convolución. La convolución es el componente básico de la CNN y por la que recibe su nombre, también es la que realiza la mayor parte de trabajo para el procesamiento. La convolución opera principalmente dos matrices, una es la matriz de entrada ya normalizada, la otra es una matriz llamada kernel ( $K$ ) o núcleo, esta matriz puede ser de  $3 \times 3$ ,  $5 \times 5$ , en general de  $k \times k$ . Con la convolución

se puede mejorar la calidad de la imagen, la nitidez, detectar bordes, entre otros aspectos. La operación de convolución entre dos funciones  $I(x, y)$  y  $K(x, y)$  de tamaño  $M \times N$  denotada por  $I(x, y) * K(x, y)$  está definida por la ecuación 2.2 en discreto-bidimensional [67].

$$C[x, y] = I[x, y] * K[x, y] = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I[m, n] K[x - m, y - n] \quad (2.2)$$

$\forall x = 1, 2, 3, \dots, \text{Ancho imagen}, y = 1, 2, 3, \dots, \text{Alto imagen}$

Donde:

$*$  : indica la operación de convolución discreta bidimensional,

$I$  : es una matriz de la imagen de entrada,

$K$  : es el kernel utilizado,

$\frac{1}{MN}$  : se normaliza al tamaño de la imagen,

En nuestro caso se trabaja con matrices cuadradas de  $k \times k$  con  $M = k, N = k$ .

**NOTA:** el kernel se puede rotar 180 grados y multiplicar cada valor del kernel por cada valor de la matriz que se desea realizar la convolución. Las coordenadas utilizadas en la ecuación 2.2 son las mismas que se utilizan al rotar el kernel.

En la figura 2.10 se muestra un ejemplo de la operación de convolución.

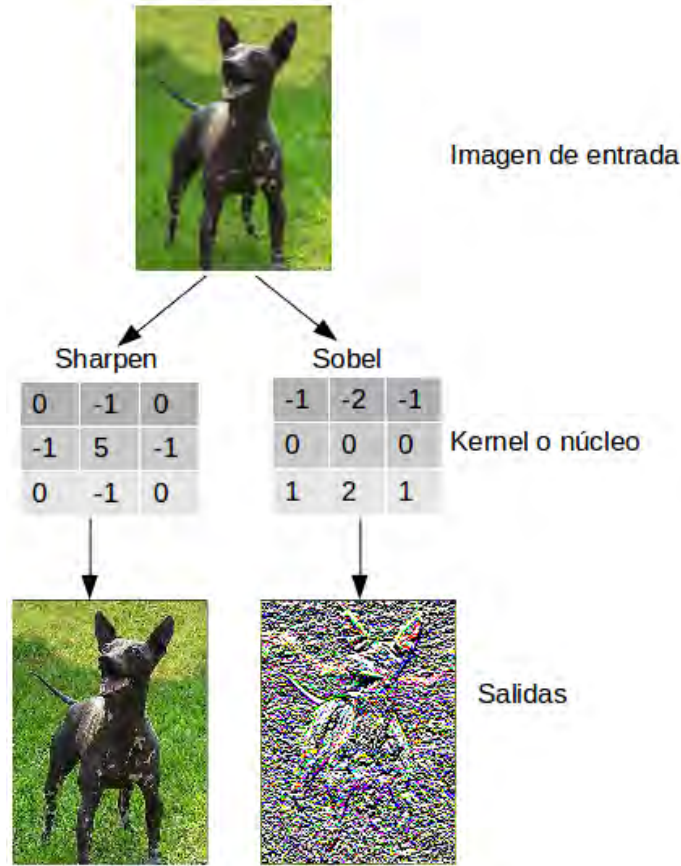
Características de una convolución:

- Utiliza un núcleo o kernel de convolución bi-dimensional cuadrado de tamaño  $k \times k$ .
- El inicio es en la esquina superior izquierda.
- La terminación es en la esquina inferior derecha.
- Existe un desplazamiento  $\rho$  píxeles tanto en horizontal como vertical.

El total de operaciones generadas en la red es importante en una matriz de  $N \times N$ , utilizando un kernel de  $k \times k$  con un relleno de  $\tau$  y un desplazamiento de  $\rho$  píxeles sobre la imagen, en el caso de la convolución se obtienen de la siguiente forma:

$$C_H = (N - k + 2\tau)/\rho + 1$$

$$C_V = (N - k + 2\tau)/\rho + 1$$



**Figura 2.10.** Ejemplo de operación de convolución.

Total de convoluciones =  $C_H \times C_V$ .

Por ejemplo en el caso de una matriz de  $N \times N$  y un kernel de  $k \times k$ , suponiendo que  $N = 227$ ,  $k = 7$ ,  $\tau = 0$  y  $\rho = 2$ , se obtiene lo siguiente:

$C_H = (227 - 7 + 2 * 0)/2 + 1 = 111$ : número de convoluciones de forma horizontal.

$C_V = (227 - 7 + 2 * 0)/2 + 1 = 111$ : número de convoluciones de forma vertical.

Total de convoluciones en una imagen =  $C_H \times C_V = (111) \times (111) = 12,321$ .

Cuando se trabaja en RGB se tendrán  $C_H \times C_V \times 3$ , con un total de 36,963 operaciones de convolución por una imagen.

Al iniciar el entrenamiento con la CNN los kernel se generan aleatoriamente, debido a que estarán cambiando en el transcurso del entrenamiento para adaptarse a las imágenes de entrada.

### (c) Pooling

La capa de pooling tiene por objetivo principal reducir la salida de la capa de convolución, de modo que la cantidad de características también sean reducidas, manteniendo la información más representativa de la imagen. Así se reduce la resolución de los atributos y los hace robustos al ruido y distorsión. El parámetro usado es una máscara  $M$  de tamaño  $k \times k$  y paso  $\rho$ . Existen diferentes tipos de pooling pero los más utilizados son.

- **Average-pooling:** obtiene el valor promedio de los elementos  $j$  de una región  $R_i$  de una imagen  $I(x, y)$  de tamaño  $N \times N$ . La podemos definir como:

$$Y_1 = \frac{1}{k \times k} \sum_{i \in [k \times k]} R_i \quad (2.3)$$

Donde:

$$R_i \in I(x, y) \quad \forall \{x, y\} \in \{1 \dots N\}$$

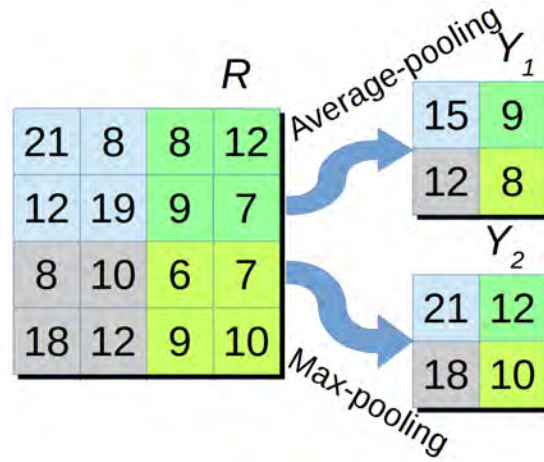
- **Max-pooling:** obtiene el valor máximo de una región  $R_i$  de tamaño  $k \times k$  de una imagen  $I(x, y)$  de tamaño  $N \times N$ . La podemos definir como:

$$Y_2 = \max\{R_i\}_{i \in [k \times k]} \quad (2.4)$$

Donde:

$$R_i \in I(x, y) \quad \forall \{x, y\} \in \{1 \dots N\}$$

Un ejemplo práctico para average-pooling y max-pooling es el que se muestra en la figura 2.11, en donde se divide la matriz  $R$  en cuatro regiones con máscara de  $2 \times 2$ , y en cada región se obtiene la media o el máximo, e. g., en la primera región podemos tomar a  $[21, 8, 12 \text{ y } 19]$ , de estos cuatro números la media es 15 tal como se muestra en la matriz dos ( $Y_1$ ) utilizando average-pooling, o en la matriz tres ( $Y_2$ ) utiliza max-pooling donde el máximo es 21.



**Figura 2.11.** Ejemplo de operación max-pooling y average-pooling sobre la matriz  $R$ .

#### (d) Activación no-lineal de la red

Para que la red funcione se necesita de la función de activación que, dada una o varias entradas, define la salida de una neurona en una red. Existen distintas funciones, siendo las más importantes la sigmoide, la tangente hiperbólica y la ReLU, las cuales se explican a continuación:

- Sigmoide, dado por la ecuación 2.5, ver figura 2.12 (a).

$$Y(x) = \frac{1}{1 + e^{-x}} \quad (2.5)$$

- Tangente hiperbólica, dado por la ecuación 2.6 y su gráfica se muestra en la figura 2.12 (b).

$$Y(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \quad (2.6)$$

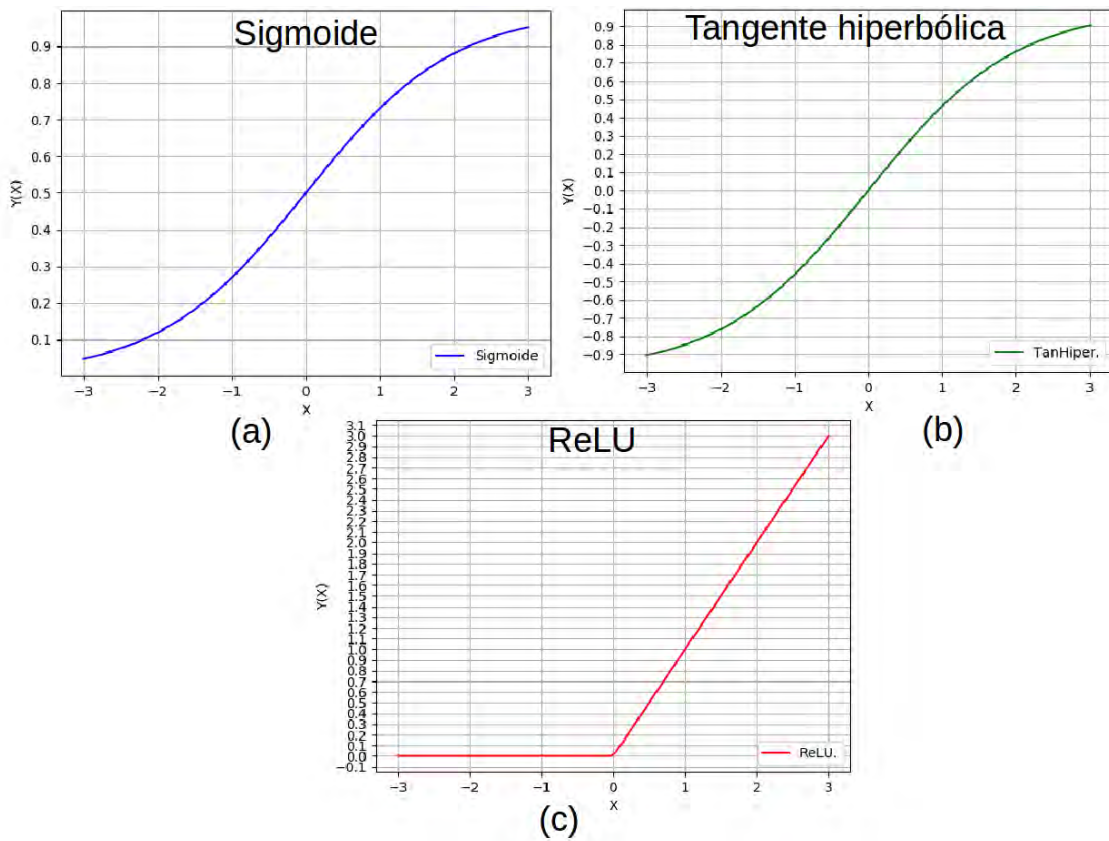
- ReLU (Rectified Linear Units): es más rápida que las funciones anteriores. En los conjuntos de imágenes grandes como ImageNet es importante ya que reduce el tiempo de cálculo y por ende, el de entrenamiento y clasificación. La dimensión de la matriz de salida  $Y$  es la misma que la matriz de entrada  $X$  de



tamaño  $N \times N$ . ReLU implementa la función descrita en 2.7, figura 2.12 (c).

$$Y(x) = \text{máx}(0, x) \quad (2.7)$$

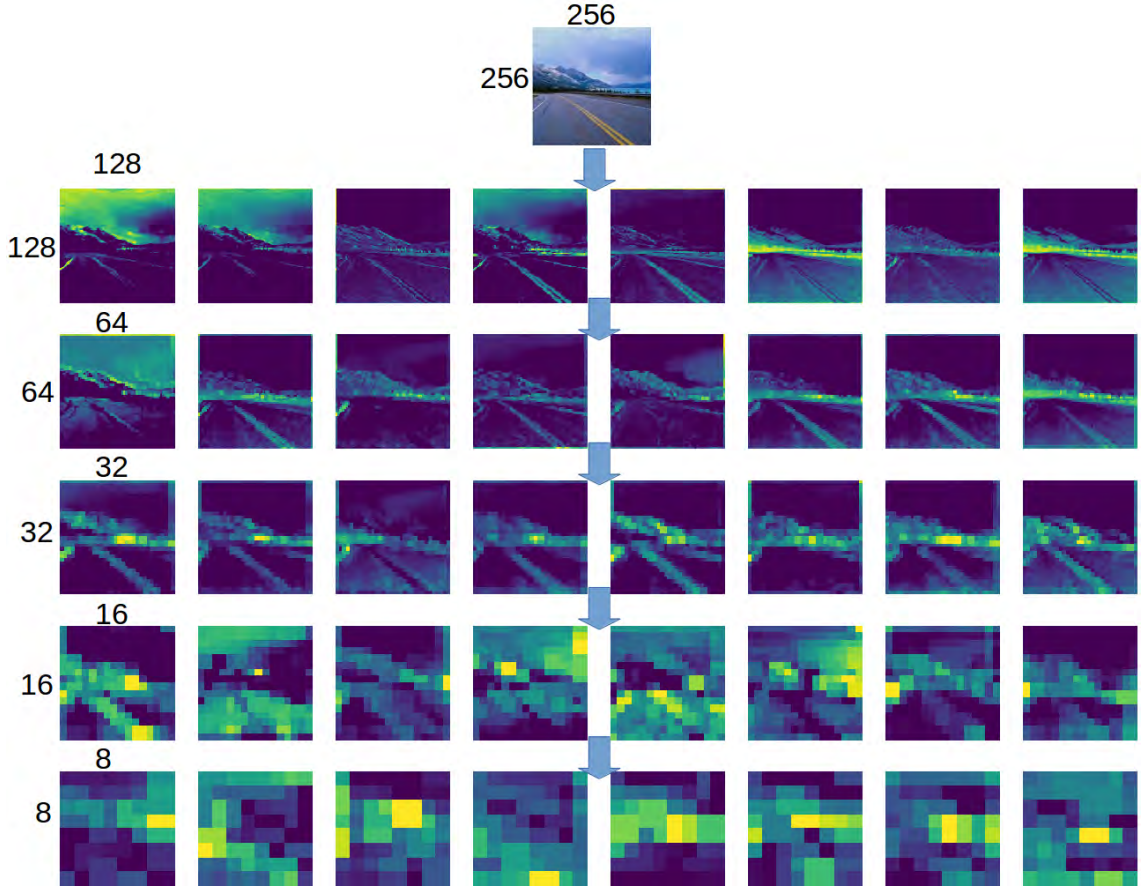
Donde:  $x \in X$



**Figura 2.12.** Gráficas de las funciones de activación: (a) sigmoide, (b) tangente hiperbólica y (c) ReLU.

Después de pasar por cinco capas de convolución, reducción y ReLU genera un resultado como el que se muestra en la figura 2.13; se describe el ingreso de una imagen de  $256 \times 256$  que se reduce hasta una de  $8 \times 8$  después de pasar por cinco diferentes capas. La salida será procesada por la capa totalmente conectada, se puede ver que existen características que representan a la imagen de entrada.





**Figura 2.13.** Resultados utilizando cinco capas, iniciando con una imagen de  $256 \times 256$  y finalizando con imágenes de  $8 \times 8$ .

## Clasificación

### (e) Capa totalmente conectada

Los resultados obtenidos en las capas anteriores se transforman a un vector unidimensional y se conectan a las capas siguientes de tal forma que las salidas de los elementos de la capa  $l - 1$  estén conectadas a cada elemento de la capa  $l$  a esto se le llama *capa totalmente conectada*, ver figura 2.14. La salida  $y^l(j)$  del elemento (neurona)  $j$  es obtenida por la siguiente función [68]:

$$y^{(l)}(j) = f^{(l)}\left(\sum_{i=1}^{N^{(l-1)}} y^{(l-1)}(i) \cdot w^{(l)}(i, j) + b^{(l)}(j)\right) \quad (2.8)$$

Donde:

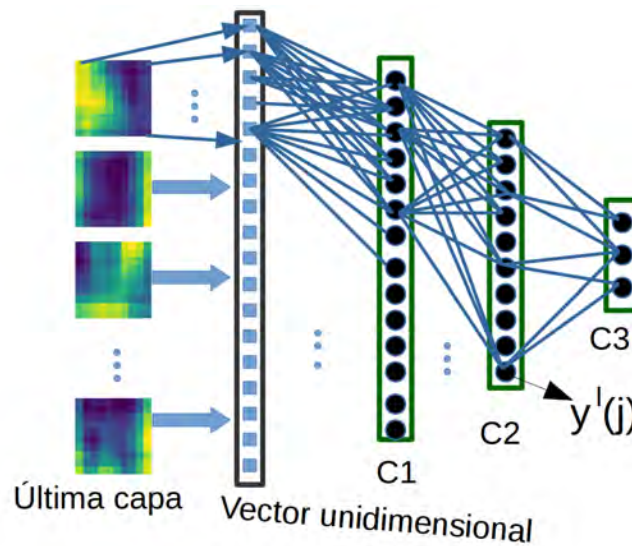
$N^{l-1}$  : es el número de elementos de la capa  $l - 1$ ,

$w^{(l)}(i, j)$  : es el peso para la conexión del elemento  $i$  en la capa  $l - 1$  al elemento  $j$  de la capa  $l$ ,

$b^{(l)}(j)$  : es el bias o valor del sesgo del elemento  $j$  en la capa  $l$ ,

$f^{(l)}$  : representa la función de activación de la capa  $l$ .

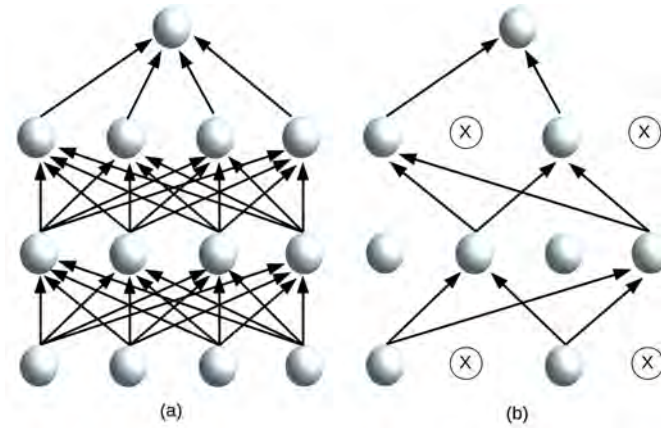
El trabajo principal de la capa totalmente conectada es clasificar los datos obtenidos en las capas anteriores. La figura 2.14 muestra un ejemplo donde se observan los resultados de la última capa de extracción de características convertidas a un vector unidimensional y procesadas en las capas totalmente conectadas.



**Figura 2.14.** Ejemplo de capas totalmente conectadas.

### (f) Dropout

*Dropout* [69] nos ayuda a desconectar un cierto número de neuronas de forma aleatoria en cada iteración del entrenamiento, así los nodos habilitados ayudan a que la red sea más específica. Es utilizada principalmente para reducir el sobre-entrenamiento, en la figura 2.15 se muestra un ejemplo.



**Figura 2.15.** En (a) se muestra la red completa, en (b) se muestra el resultado del dropout.

### (g) Softmax

*Softmax* [70] es una función que se utiliza generalmente en la última capa totalmente conectada para predecir la imagen de entrada con un valor entre 0 y 1. Está definida por la siguiente función:

$$s_j(x) = \frac{e^{x_j}}{\sum_{k=1}^n e^{x_k}} \forall j = 1, \dots, n \quad (2.9)$$

Donde:

$x_j$  : es el valor de un elemento de la última capa totalmente conectada.

$n$  : es el número de elementos de la capa de salida.

Todos los valores de salida están entre cero y uno y la suma es igual a uno.

La función de pérdida para softmax intenta minimizar la entropía cruzada entre las clases estimadas y las clases reales. La función de pérdida de la entropía cruzada se define por la siguiente función:

$$L_i = -\log\left(\frac{e^{T_{y_i}}}{\sum_j e^{T_j}}\right) \quad (2.10)$$

Donde:

$T_j$  : corresponde a un elemento  $j$  del vector de clases  $T$ ,

$L_i$  : es la pérdida para el conjunto de datos de los ejemplos de entrenamiento junto

con un término de regularización.

### (h) Capa de salida

El trabajo principal de esta capa es mostrar el resultado del análisis hecho por las capas anteriores, donde muestra, que la probabilidad de éxito de que la imagen de entrada sea un perro es de 90 %, o un gato con probabilidad de 10 %. Es importante recalcar que pueden ser varios los resultados de salida con diferentes probabilidades de éxito, esto depende de la arquitectura de la CNN.

Refiriendo a la estructura general de una CNN (figura 2.7) remarcaremos que, en general, las primeras capas extraen características mientras que las dos últimas codifican la etapa de clasificación. En la CNN se inicia con la entrada directa de datos de la imagen tal cual como matriz, después se le hace una operación de convolución utilizando un kernel, este depende de las arquitecturas propuestas, al realizar la operación se obtienen diferentes matrices. A cada matriz resultante se le aplica una operación de reducción obteniendo las características más representativas de cada matriz. Al final se utiliza un método de clasificación (capas completamente conectadas) para dar como salida la probabilidad de que la imagen de entrada sea, por ejemplo, un perro con probabilidad de éxito de 90 % o que sea gato de 10 %, entre otros casos.

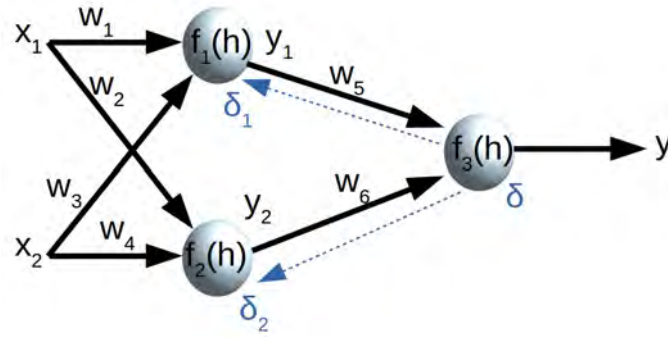
## Métodos de aprendizaje

### Backpropagation

*Backpropagation* es una técnica utilizada en las redes neuronales para propagar errores a través de la arquitectura y para adaptar los pesos [71]. El entrenamiento de una red neuronal con Backpropagation se compone de dos pasos simples, propagación hacia adelante y propagación hacia atrás. El primero se refiere a la utilización de la red para realizar la clasificación. El segundo hace el cálculo del error de clasificación y se propaga a través de la red neuronal hacia atrás. Los

pesos se actualizan en función del error, la tasa de aprendizaje y el gradiente de la activación.

Como ejemplo se muestra una red de tres elementos en la figura 2.16. De esta red ingresan dos elementos  $x_1, x_2$  pasando por la red en  $f_1(h), f_2(h), f_3(h)$  respectivamente con diferentes pesos  $w_1, w_2, w_3, w_4$ . Al final la red da una salida  $y$ , esta se valida con un valor esperado  $z$  para saber si es correcto. También tiene una tasa de aprendizaje  $\alpha$ .



**Figura 2.16.** Red utilizada como ejemplo para el Backpropagation.

Primero se hacen los cálculos de  $y_1, y_2, y$  que serán los valores de salida de cada elemento. Esto de la siguiente forma:

$$y_1 = f_1(w_1(x_1) + w_3(x_2))$$

$$y_2 = f_2(w_2(x_1) + w_4(x_2))$$

$$y = f_3(w_5(y_1) + w_6(y_2))$$

Después se propaga el error hacia atrás de la siguiente forma:

$$\delta = z - y$$

$$\delta_1 = w_5(\delta)$$

$$\delta_2 = w_6(\delta)$$

Para finalmente obtener los pesos actualizados:

$$w'_1 = w_1 + \alpha \delta_1 \frac{df_1(h)}{dh} x_1$$

$$w'_2 = w_2 + \alpha \delta_2 \frac{df_2(h)}{dh} x_1$$

$$w'_3 = w_3 + \alpha \delta_1 \frac{df_1(h)}{dh} x_2$$

$$w'_4 = w_4 + \alpha \delta_2 \frac{df_2(h)}{dh} x_2$$

$$w'_5 = w_5 + \alpha \delta \frac{df_3(h)}{dh} y_1$$

$$w'_6 = w_6 + \alpha \delta \frac{df_3(h)}{dh} y_2$$

### Gradiente descendente

En las redes neuronales el *gradiente descendente* utiliza el Backpropagation para el aprendizaje, en las redes neuronales convoluciones pueden ser utilizados: el gradiente descendente estocástico [72], gradiente descendente estocástico con momentos [73], ADAM [74], entre otros [72]. El gradiente descendente es una forma de minimizar la función objetivo  $J(\theta)$ . Actualizando los parámetros en la dirección opuesta del gradiente de la función objetivo  $\nabla_{\theta} J(\theta)$ . La función del gradiente descendente se obtiene de la siguiente forma:

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t) \quad (2.11)$$

Donde:

$\alpha$  : es la tasa de aprendizaje,

$\theta_t$  : es el punto de inicio,

$\theta_{t+1}$ : es el siguiente punto a encontrar,

$-\nabla J(\theta_t)$  : es la dirección negativa del gradiente o la derivada direccional de  $J(\theta_t)$ .

### Gradiente descendente estocástico

Gradiente descendente estocástico o stochastic gradient descent (SGD) realiza una actualización por cada ejemplo de entrenamiento  $x^{(i)}$  y  $y^{(i)}$  y está definida por la siguiente función:

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t; x^{(i)}; y^{(i)}) \quad (2.12)$$

Donde:

$\alpha$  : es la tasa de aprendizaje,

$\theta_t$  : es el punto de inicio,

$\theta_{t+1}$ : es el siguiente punto a encontrar,

$x^{(i)}, y^{(i)}$  : ejemplos de entrenamiento,  
–  $\nabla J(\theta_{t-1}; x^{(i)}; y^{(i)})$  : es la dirección negativa del gradiente.

## Gradiente descendente estocástico con momentos

El *gradiente descendente estocástico con momentos* es una técnica para acelerar el gradiente descendente estocástico, evita el sobreajuste y busca posibilidades para parar aplicando el momento [75]. Se obtiene con la siguiente función:

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t) + \gamma(\theta_t - \theta_{t-1}) \quad (2.13)$$

Donde:

$\alpha$  : es la tasa de aprendizaje,

$\theta_t$  : es el punto de inicio,

$\theta_{t+1}$ : es el siguiente punto a encontrar,

–  $\nabla J(\theta_t)$  : es la dirección negativa del gradiente,

$\gamma$  : determina la diferencia del gradiente anterior con el gradiente actual.

## ADAM (Adaptive Moment Estimation)

*ADAM* [74] es un algoritmo de optimización, es usado principalmente en redes neuronales convolucionales, hace referencia a Estimación del Momento Adaptativo o (Adaptive Moment Estimation). Calcula las tasas de aprendizaje adaptativo para cada parámetro y está basada en el gradiente. Es eficiente para grandes bases de datos y gran variedad de parámetros logrando menor tiempo de aprendizaje, es sencillo y eficiente en el procesamiento de datos, también tiene sus variaciones. La función es la siguiente:

$$\theta_t = \theta_{t-1} - \frac{\alpha \cdot m_t}{\sqrt{v_t} + \epsilon} \quad (2.14)$$

Donde:

- $\alpha = 0.001$

- $\epsilon = 10^{-8}$
- $m_t$ : Calcula la estimación del primer momento con el sesgo corregido.
- $v_t$ : Calcula la estimación del segundo momento con el sesgo corregido.

## Evaluación de desempeño

La evaluación del desempeño de las redes neuronales convolucionales la podemos enfocar en la optimización (exactitud) en el entrenamiento y la generalización en las pruebas. La optimización se refiere al proceso de ajuste de la red para obtener el mejor rendimiento posible con los datos de entrenamiento. La generalización se refiere a qué tan bien funciona la red entrenada con datos que nunca ha visto como son los de pruebas. Para esto influyen los conjuntos de imágenes, algoritmos de aprendizaje, software, recursos de cómputo utilizados, entre otras variables.

Primero se separa un conjunto de imágenes para entrenamiento y otra para pruebas (se puede separar una tercera de validación para afinar los pesos). Tradicionalmente se selecciona el 70 % de los datos de origen para formar el conjunto de entrenamiento y el 30 % para el conjunto de prueba. Después se entrena la red y finalmente se evalúa con las imágenes no aprendidas. A continuación, primero describimos el entrenamiento, después la validación y finalmente las pruebas [76].

## Entrenamiento

El procedimiento utilizado para llevar a cabo el proceso de aprendizaje de una CNN se denomina entrenamiento. Como vimos anteriormente, existen diferentes algoritmos de aprendizaje que pueden resultar interesantes ya que presentan distintas características y rendimiento que los pueden hacer más o menos adecuados dependiendo de las características del problema concreto a resolver.

El problema del aprendizaje se formula en términos de la minimización de una función de error asociada, la cual está compuesta por dos términos: uno que se denomina término de error, el cual evalúa cómo se ajusta la salida de la red al conjunto de datos de entrenamiento y otro que se denomina término de



regularización que se utiliza para evitar el sobreaprendizaje [77].

La función de error depende por completo de los parámetros de la red: pesos y bias. El objetivo del aprendizaje es encontrar los valores de los pesos para los que se obtiene un mínimo global de la función de error, convirtiendo el problema de aprendizaje en un problema de optimización [78].

Obtener un buen rendimiento en entrenamiento no implica que en las pruebas se obtenga una buena precisión. El tema a considerar es el sobre-ajuste o sobre-entrenamiento, se da por que los valores de los parámetros se ajustan demasiado bien a la especialidad de los ejemplos de entrenamiento sesgados y son óptimos en el sentido de minimizar el error de generalización dado por la función de pérdida [79].

Existen conceptos que es necesario conocer para realizar un entrenamiento, tales como:

- **Lote:** cuando se entrena una red se tiene que ingresar las imágenes en ella, puede ser de una por una o varias. La desventaja de hacerlo de una por una es que no se utilizan todos los recursos disponibles, por lo tanto el tiempo de procesamiento de datos es largo. La otra forma es ingresar varias imágenes y procesarlas en paralelo dependiendo de los recursos que se tengan.

Al entrenar se define una cantidad de imágenes o “lotes” (batch) [80] de manera aleatoria extraída de la base de datos de entrenamiento. Por ejemplo en la primera iteración se puede extraer un lote de 32 imágenes de la base de datos para aun primer análisis, estas imágenes pasan por la red modificando los pesos y los bias, en la siguiente iteración se vuelven a extraer otras 32 imágenes de manera aleatoria, así hasta completar un número de iteraciones. La cantidad de lotes depende de la cantidad de memoria RAM que se tenga en la CPU o GPU, si la cantidad de información sobrepasa la memoria no podrá entrenar la red.

- **Época:** una época se completa cuando todo el conjunto de entrenamiento pasa por la red de aprendizaje, por ejemplo: si se tienen 20,000 imágenes para entrenamiento y el lote es igual a 32, entonces, una época se cumple cuando hay

625 iteraciones ya que cada lote es una iteración se obtendría de la siguiente forma  $(20,000/\text{NumeroDeLote})$  [80].

- **Tasa de aprendizaje:** la tasa de aprendizaje es uno de los parámetros más importantes, influye en que la red aprenda de manera rápida o lenta, se va disminuyendo cuando el error de validación ya no cambia, se dan mejores resultados con el decaimiento lineal [66].
- **Batch Normalization:** cuando se entrena una red, la entrada de datos de las capas no son iguales, esto hace que en las capas superiores se obtengan salidas diferentes, el aprendizaje será lento debido a que se tienen que adaptar los datos bajando la tasa de aprendizaje. *Batch Normalization* (BN) [81] permite normalizar cada lote de entrenamiento, con esto, se puede utilizar tasas de aprendizaje más altas, y reducir el tiempo de entrenamiento.

## Validación

El conjunto de datos de validación sirve para afinar los pesos con nuevas imágenes después del entrenamiento. Una vez validado se puede evaluar el rendimiento con el conjunto de datos de prueba. Es importante decir que en el entrenamiento y la validación la red sigue aprendiendo y son necesarias las imágenes con su clasificación [82].

## Prueba

Una vez que terminaron las épocas de aprendizaje en el entrenamiento podemos evaluar la precisión con el conjunto de datos de pruebas. Las pruebas se realizan para validar que tan bien aprendió la red. En las pruebas se evalúa la diferencia entre las imágenes que se clasificaron de forma correcta y las que no.

Para evitar evaluar en cada época de entrenamiento se puede realizar un análisis de los resultados de entrenamientos para ver a partir de que época la red obtuvo mejor rendimiento [66].

Una herramienta muy útil para realizar la evaluación de la precisión es la matriz de confusión. De esta matriz se puede obtener de la diagonal el éxito total, el

porcentaje de clasificaciones correctas, y el porcentaje de las esperadas.

En resumen:

- El conjunto de datos de entrenamiento obtiene los pesos y ajusta los parámetros en la red.
- El conjunto de datos de validación afina los pesos con datos y se puede utilizar para comparar el diseño entre distintas arquitecturas.
- El conjunto de datos de prueba verifica que tan capaz es la red de predecir la clasificación de la imagen de entrada.

Para realizar los entrenamientos se necesita de hardware y software especializado. El hardware para realizar múltiples operaciones de manera paralela y el software para realizar la construcción e implementación de las redes, como se describe a continuación.

## Hardware

Existen diferentes formas de realizar entrenamiento de forma paralela, tales como: utilizar un cluster de computadoras o utilizar una sola computadora integrando dispositivos como las GPUs. Utilizar una u otra depende de los recursos con que se cuentan. En este trabajo se utilizaron GPUs, se describen sus características a continuación.

## GPUs

La eficiencia en el aprendizaje de una red neuronal depende del entrenamiento pero el hardware con que se cuenta influye en el tiempo para obtener los resultados. Entrenar una red con un conjunto de miles de imágenes puede tardar horas, días, semanas o incluso meses usando solo la CPU (también depende de la profundidad de la red). Para disminuir el tiempo se utilizan unidades de procesamiento gráfico o Graphics Processing Unit (GPU) con miles de núcleos que realizan operaciones de forma paralela. Las redes neuronales han cobrado fuerza al poder ser entrenadas con este hardware para el procesamiento de datos, así las tareas de pre-procesamiento,

extracción de características y clasificación se dividen de forma paralela.

NVIDIA[83] es una empresa que vende GPUs, proporciona los drivers, también CUDA y CUDA toolkit para poder utilizarlas. Las GPUs se pueden diferenciar por su capacidad de cómputo, número de núcleos, memoria, arquitectura, entre otras.

El entrenamiento de una CNN representa un alto costo computacional [84] [68], por lo que las GPUs son utilizados como motores de cálculo que tienen numerosas unidades de ejecución y gran ancho de banda de memoria para obtener un alto rendimiento computacional. Los entrenamientos utilizando GPUs pueden reducirse de días a horas, dependiendo del hardware que se cuente. En la figura 2.17 se puede ver un ejemplo de una GPU NVIDIA TITAN X.

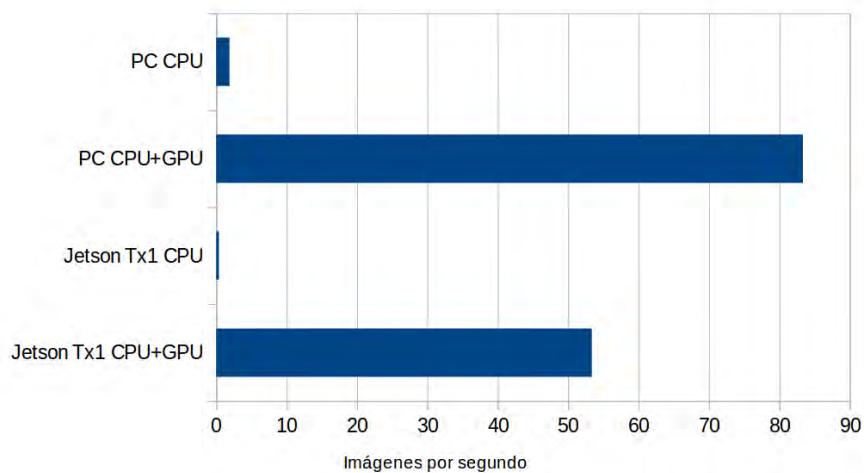


**Figura 2.17.** Ejemplo de GPU, NVIDIA TITAN X [7].

Existen dispositivos como la tarjeta NVIDIA jetson TX1 que traen incluido una GPU para mejorar el tiempo de procesamiento de datos. En el trabajo [8] hacen una comparación entre una computadora con un procesador intel Core i7-5930K a 3.50 GHz con 6 Cores usando la GPU GeForce GTX 970 con 1664 CUDA cores y la Jetson TX1 que tiene una GPU Maxwell GPU con 256 cores. En la figura 2.18 se puede ver los resultados.

## Software

Para utilizar los GPUs se necesita del driver del dispositivo. A un nivel más arriba está CUDA para poder controlar el dispositivo, éste puede ser utilizado por diferentes ambientes de trabajo o frameworks que trabajan con aprendizaje profundo. Como



**Figura 2.18.** Rendimiento de GoogleNet con la comparación usando CPU-GPU [8].

ejemplo tenemos a TensorFlow <sup>1</sup> y Caffe <sup>2</sup>, aunque no son los únicos, son los más destacados [83].

## CUDA

CUDA (Compute Unified Device Architecture) [85] es una arquitectura de cálculo paralelo de NVIDIA que aprovecha la potencia de las GPUs. Está pensado para realizar coprocesamiento repartido entre la CPU y la GPU. La parte secuencial de cálculo se ejecuta en la CPU, la parte paralela del cálculo se ejecutan en miles de núcleos de la GPU. CUDA se basa en la programación en hilos ejecutados concurrentemente, los bloques de hilos se agrupan en Grids para ejecutarse en el kernel. Se puede utilizar para la programación con C, C++, Python o en MATLAB.

## TensorFlow

*TensorFlow* [18] es una biblioteca de software libre es utilizada para realizar cálculos numéricos utilizando grafos para representar el proceso que sigue la programación. Los nodos representan las operaciones matemáticas, las aristas son los llamados tensores que, principalmente, son matrices de datos multidimensionales. Con los

<sup>1</sup>TensorFlow. <https://www.tensorflow.org>, (Enero 2017) .

<sup>2</sup>Caffe. <http://caffe.berkeleyvision.org/>, (Enero 2017) .

tensores se pueden representar las entradas y las salidas de las operaciones realizadas en las redes neuronales [86].

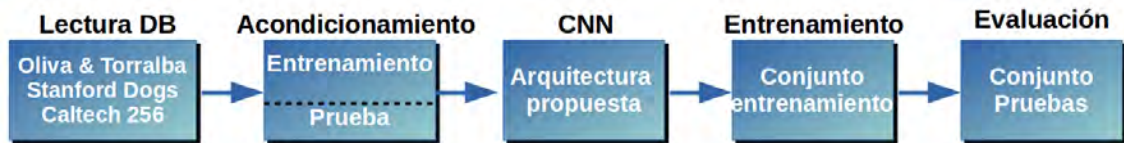
Se puede utilizar en una o varias computadoras, también se pueden utilizar las GPUs o para dispositivos móviles. Google desarrolló esta librería para la construcción e implementación de las CNNs y otras redes. Fue liberado TensorFlow para que sea un proyecto de código abierto, se ha utilizado ampliamente para la investigación de aprendizaje automático.

En el trabajo de investigación [87] se muestra una comparación con otras librerías como Theano, Torch, y Caffe.

# Capítulo 3

## Metodología

La metodología que seguimos fue, como primer paso, seleccionar, los conjuntos de imágenes (públicos) que serán utilizados. Después, separamos las imágenes en entrenamiento y prueba. Posteriormente, se realizó el diseño y la construcción de la red tomando como base el diseño de AlexNet, GoogleNet y ResNet. Finalmente, entrenamos y validamos la red diseñada. En la figura 3.1 se muestra la metodología seguida para el desarrollo de esta investigación.



**Figura 3.1.** Metodología de investigación para la construcción, entrenamiento y prueba de la red.

### 3.1. Conjuntos de datos utilizados

En este trabajo utilizamos tres conjuntos de imágenes diferentes que tienen como características principales: categorías con un número de imágenes desigual e imágenes con tamaños diferentes. Estas se describen brevemente a continuación:

- **Oliva & Torralba** [19]: este conjunto de datos consta de 2,688 imágenes a color pertenecientes a ocho categorías o clases; las imágenes fueron obtenidas de diferentes fuentes: bases de datos comerciales, sitios web y cámaras digitales.
- **Stanford Dogs** [20]: este conjunto de datos consta de 20,580 imágenes a color pertenecientes a 120 clases o razas de perros de todo el mundo; las imágenes

fueron obtenidas de la base de datos ImageNet [23], el tamaño de cada imagen es variable.

- **Caltech 256** [21]: este conjunto de datos consta de 30,607 imágenes a color pertenecientes a 257 categorías, el número mínimo de imágenes en cualquier categoría es de 80.

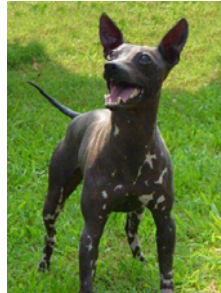
En la tabla 3.1 se muestran las características de los conjuntos de datos, también en las figuras 3.2, 3.3 y 3.4 se muestran algunas imágenes referente a cada base de datos.

**Tabla 3.1.** Conjuntos de datos utilizados y sus características.

Conjunto de datos	Clases	Dimensiones			No. imágenes		
		Ancho	Alto	Prof	Total	Min x clase	Max x clase
Oliva & Torralba	8	256	256	3	2,688	260	410
Stanford Dogs	120	200 – 500	150 – 500	3	20,580	148	252
Caltech 256	257	300	200	3	30,607	80	827



**Figura 3.2.** Imágenes Oliva & Torralba.



**Figura 3.3.** Imágenes Stanford Dogs.



**Figura 3.4.** Imágenes Caltech 256.

## 3.2. Acondicionamiento de los conjuntos de datos para entrenamiento y prueba

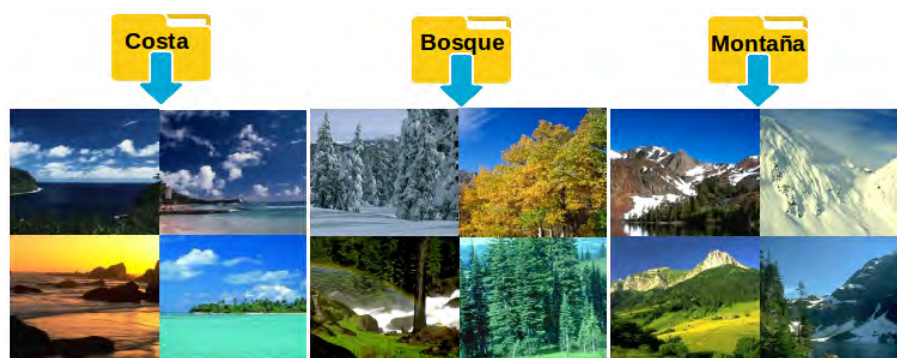
Existen diferentes formas de ingresar los datos a la red, tales como: cargar los datos en cada paso de ejecución, precargar los datos en variables y generar archivos que contengan las imágenes para después cargarlas. Cargar las imágenes en cada paso de ejecución tiene el problema de que es lento el proceso de lectura para grandes cantidades de datos. Precargar los datos en variables, solo se utiliza para pequeños



conjuntos de datos, ya que hay un límite de carga en memoria lo que implica sobrecargarla; generando problemas en los procesos. El método más utilizado para el manejo de grandes cantidades de imágenes es la generación de archivos que contienen las imágenes para después cargarlas; razón por la cual es el método seleccionado para llevar a cabo los experimentos.

Estos archivos contienen varias imágenes ya pre-procesadas y con las etiquetas de la clase a la que pertenece. Para realizar este trabajo se debe de tomar en cuenta lo siguiente:

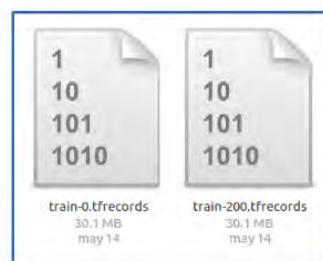
- Mismo tamaño de la imagen. Todas las imágenes deben tener el mismo tamaño para que puedan ser ingresadas a la red, por ejemplo  $224 \times 224 \times 3$  píxeles. Entre más grande es la imagen la cantidad de operaciones a realizar aumenta, se debe tener cuidado con la definición del tamaño.
- Mismo espacio de color. El espacio de color RGB es el más utilizado en este tipo de redes porque ha dado buenos resultados como ya se ha mencionado.
- Imágenes ya clasificadas. Se necesitan imágenes clasificadas, por ejemplo todas las montañas en una carpeta, las ciudades en otra, donde cada carpeta representa a una clase diferente. El contenido de las carpetas se muestran en la figura 3.5 del conjunto de imágenes Oliva & Torralba [19].



**Figura 3.5.** Ejemplos de imágenes de la clase costa, bosque y montaña.

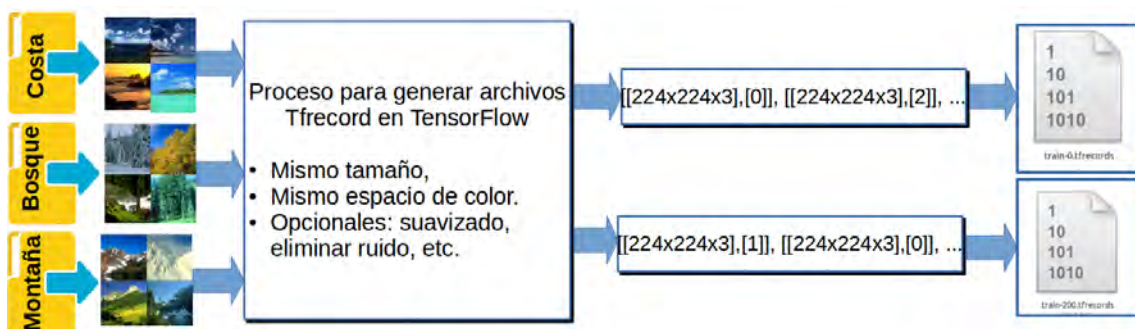
Definidas las características que tendrá la imagen, se puede proceder a crear el archivo que contendrá las imágenes. Estos archivos pueden contener todas las imágenes o con un número específico, se debe tener presente la cantidad de memoria que se tiene en la GPU y la CPU para no tener problemas.

Al generar estos archivos podemos dividir la base de datos en un conjunto de entrenamiento y un conjunto de prueba. Tradicionalmente el conjunto de entrenamiento se forma con el 70 % del conjunto total, mientras que el 30 % restante se utiliza para formar el conjunto de prueba. Ejemplo de estos archivos se puede ver en la figura 3.6. Los datos son muestreados de forma aleatoria para asegurar que los conjuntos sean representativos de todo el conjunto de datos y para eliminar el sesgo de selección, lo que puede minimizar los efectos de las diferencias entre los datos y determinar las características más significativas.



**Figura 3.6.** Archivos que contienen imágenes, listos para ser ingresados a la red.

En general, el proceso de creación de archivos se muestra en la figura 3.7. Es importante hacer notar que la clase se convierte a un valor numérico debido a que en la capa totalmente conectada se hacen operaciones para detectar a que clase pertenece la imagen.



**Figura 3.7.** Proceso para generar archivos tfrecord en TensorFlow.

Para generar estos archivos desarrollamos un toolbox que se describe en el apéndice B, este separa conjuntos de imágenes para entrenamiento, validación y pruebas, dependiendo del porcentaje seleccionado. También, genera un archivo con

**Tabla 3.2.** Conjuntos de imágenes separadas en entrenamiento y pruebas utilizando el 70 % de imágenes por clase para entrenamiento y el 30 % para pruebas.

Conjunto de datos	Clases	Cantidad de imágenes		
		Entrenamiento	Prueba	Total
Oliva & Torralba	8	1,879	809	2,688
Stanford Dogs	120	14,358	6,222	20,580
Caltech 256	257	21,314	9,293	30,607

valores que serán utilizados al momento del entrenamiento, como son: el tamaño de la imagen, el número de clases, el total de imágenes, entre otros.

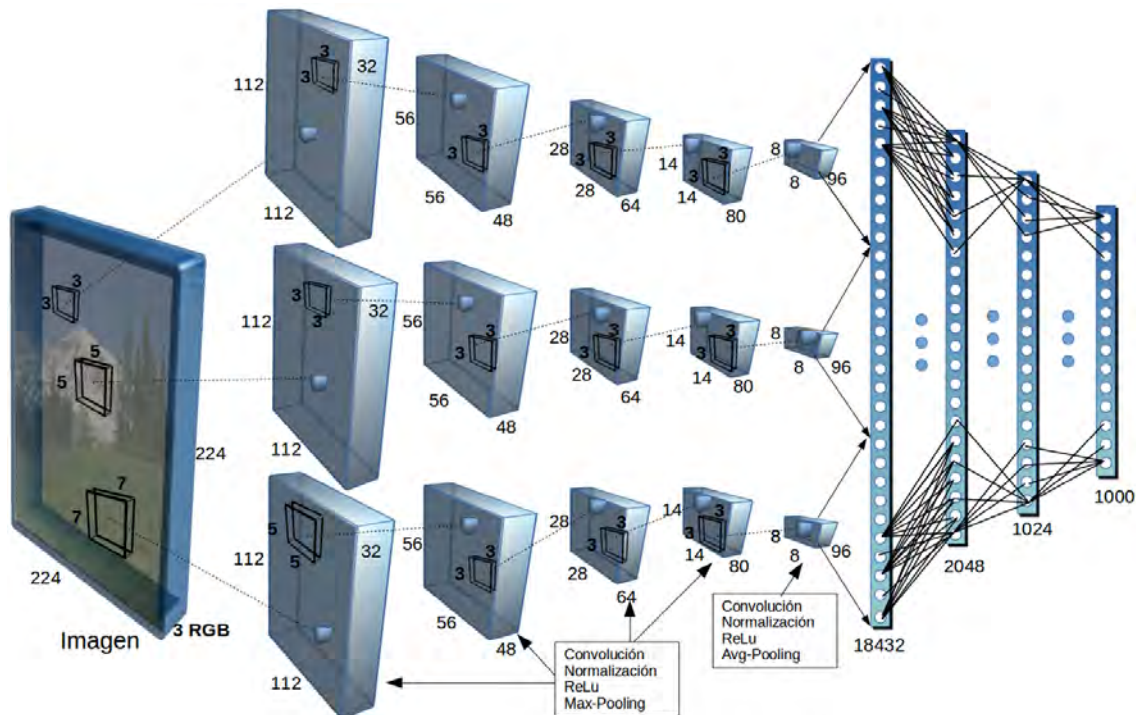
En la tabla 3.2 se muestran las características de cada conjunto de datos ya separadas en entrenamiento y pruebas, estos fueron utilizados para realizar los experimentos en este proyecto.

### 3.3. Arquitectura propuesta

Para la construcción de una nueva arquitectura de red neuronal convolucional se necesita recopilar información en diferentes aspectos. Sabemos que las redes neuronales convolucionales están formadas por neuronas que tienen pesos y sesgos, estos son guardados y utilizados para que la red aprenda. El tamaño de la imagen influye en la red diseñada para la extracción de características; se puede iniciar con una imagen de  $224 \times 224 \times 3$ , al ir pasando por varias capas se van extrayendo los parámetros más representativos. En las capas de extracción de características finales el vector multidimensional puede ser muy pequeño, por ejemplo de  $6 \times 6 \times (\text{número de filtros})$ ; este vector se transforma a un vector unidimensional para poder realizar operaciones con la capa totalmente conectada. Finalmente, se realiza la clasificación reduciendo las neuronas al número de clases del que se está haciendo el estudio.

Se han realizado pruebas con imágenes de diferentes tamaños, mientras sea más grande la imagen se extraen más características, pero con mayor cantidad de procesamiento de datos. Se decidió utilizar como entrada a la red una imagen cuadrada de  $224 \times 224$  y con una profundidad de tres (RGB) con la finalidad de poder comparar los resultados con otros modelos famosos como AlexNet, GoogleNet o ResNet.

La arquitectura propuesta se muestra en la figura 3.8, la cual está compuesta de dieciocho capas de las cuales, en la parte de extracción de características se encuentran quince capas divididas en tres secciones (cinco capas cada una). En la parte de clasificación se encuentran tres capas totalmente conectadas.



**Figura 3.8.** Diseño de red neuronal convolucional.

Una vez analizada y diseñada la red, procedemos a construirla utilizando Python 2.7 y TensorFlow 1.7.

## 3.4. Construcción de la red

En las siguientes secciones se describe la forma en que se construye la red. Primero el bloque de extracción características y después el de clasificación.

### Extracción de características

Para la construcción de la red se crearon funciones como convolución, batch normalization, entre otras. Estas son llamadas para reutilizar el código.

**Primera red con un kernel de  $3 \times 3$** 

La primera red que se codifica es la que utiliza un kernel de  $3 \times 3$  en todas sus capas de convolución. De la primera a la cuarta capa utiliza un max-pooling, en la quinta capa utiliza un average-pooling, en todas las capas utiliza convolución y normalización por lotes, como se puede ver en el algoritmo 1. La red se incrementa a razón de 16 filtros por capa debido a que se va perdiendo información de la imagen a medida que la red es más profunda, agregando filtros se puede recuperar parte de la información.

---

**Algoritmo 1** Red con kernel de  $3 \times 3$ 

---

**Entrada:** network3x3=image, NumFiltros=32, NucleoConv=3, NumCapas=1**Salida:** network3x3

```
1: mientras NumCapas<6 hacer
2:   network3x3 = Convolucion(network3x3, NucleoConv, NumFiltros)
3:   network3x3 = batchNormalization(network3x3)
4:   network3x3 = relu(network3x3)
5:   si NumCapas=5 entonces
6:     network3x3 = avgPool(network3x3)
7:   si no
8:     network3x3 = maxPool(network3x3)
9:   fin si
10:  NumFiltros=NumFiltros+16
11:  NumCapas=NumCapas+1
12: fin mientras
13: devolver network3x3
```

---

**Segunda red con un kernel de  $5 \times 5$** 

La segunda red utiliza un kernel de  $5 \times 5$  en la primera capa de convolución, en la siguiente capa se cambia a una de  $3 \times 3$ . Cabe destacar que en las pruebas esta red es la que da mejor desempeño, puede ser utilizada sin necesidad de agregar las otras tres redes si no se cuenta con un equipo de cómputo potente para entrenamiento. Se puede ver el diseño en el algoritmo 2.

---

**Algoritmo 2** Red con kernel de  $5 \times 5$ 

---

**Entrada:** network5x5=image, NumFiltros=32, NucleoConv=5, NumCapas=1

**Salida:** network5x5

```
1: mientras NumCapas<6 hacer
2:   network5x5 = Convolucion(network5x5, NucleoConv, NumFiltros)
3:   network5x5 = batchNormalization(network5x5)
4:   network5x5 = relu(network5x5)
5:   si NumCapas=2 entonces
6:     intNucleoConv=3
7:   fin si
8:   si NumCapas=5 entonces
9:     network5x5 = avgPool(network5x5)
10:  si no
11:    network5x5 = maxPool(network5x5)
12:  fin si
13:  NumFiltros=NumFiltros+16
14:  NumCapas=NumCapas+1
15: fin mientras
16: devolver network5x5
```

---

#### Tercera red con un kernel de $7 \times 7$

La tercera red utiliza un kernel de  $7 \times 7$  en la primera capa de convolución, en la siguiente capa se cambia a una de  $5 \times 5$  y en la tercera se cambia  $3 \times 3$ . El código es parecido al de las anteriores redes.

#### Concatenación de las tres redes

Finalmente, se concatenan las tres redes generadas anteriormente, haciendo notar que cada una extrae diferente información dependiendo del kernel, la forma como se realizó se describe en la siguiente línea utilizando TensorFlow.

$$network = Concatenacion([network3x3, network5x5, network7x7])$$

Se han realizado pruebas con cada una de las redes y también de forma combinada. Con las tres juntas se obtuvieron los mejores resultados en cuanto a rendimiento. También se hicieron pruebas con el kernel de  $11 \times 11$  pero disminuyó el rendimiento y también aumentó el tiempo de entrenamiento.

## Clasificación

Para las capas totalmente conectadas se realiza lo siguiente:.

---

**Entrada:** Requiere el resultado de la concatenación de las redes (*network*)

**Salida:** network

```

1: network = flatten(network)
2: network = PrimeraCapaTotalmenteConectada(network, 2048)
3: network = relu(network)
4: network = batchNormalization(network)
5: network = dropout(network, dropout)
6: network = SegundaCapaTotalmenteConectada(network, 1024)
7: network = relu(network)
8: network = batchNormalization(network)
9: network = dropout(network)
10: network = TerceraCapaTotalmenteConectada(network, NumeroClases)
11: network
12: devolver network

```

---

El diseño general de la red se muestra en la figura 3.9 utilizando Tensorboard para visualizarlo de forma gráfica.

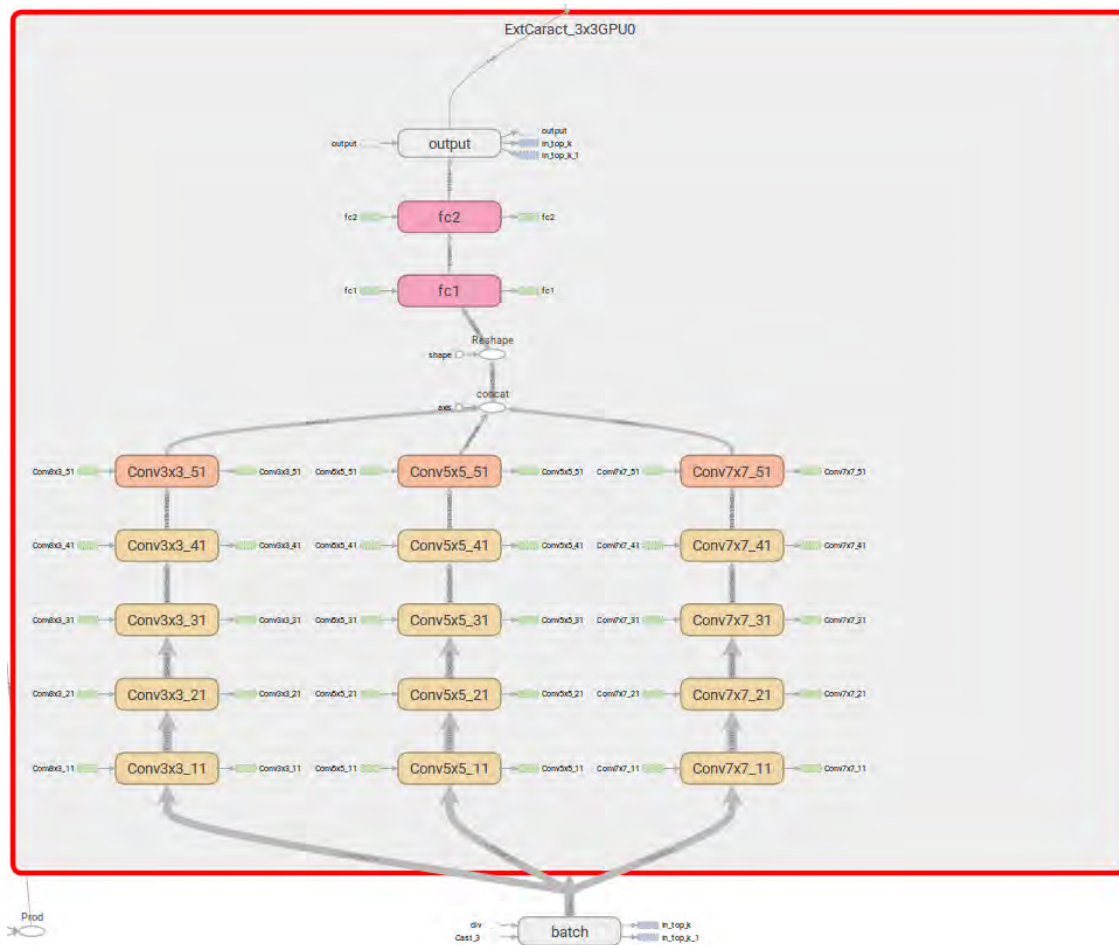
La red se comparó realizando redes independientes de cada kernel y una combinación entre ellas obteniendo mejores resultados con la red diseñada pero con mayor tiempo de entrenamiento, como se muestra en la figura 3.10. Se entrenó con 250 épocas cada una con los mismos parámetros de aprendizaje. En la tabla 3.3 se muestran los mejores resultados.

**Tabla 3.3.** Comparación con la red diseñada y redes usadas de diferentes kernel.

Épocas	Redes con diferentes kernel						
	3x3	5x5	7x7	3x3_5x5	3x3_7x7	5x5_7x7	3x3_5x5_7x7
210	91.84 %	91.34 %	92.95 %	91.71 %	91.59 %	92.21 %	93.44 %
220	92.21 %	91.96 %	92.70 %	92.21 %	93.07 %	92.70 %	92.45 %
230	91.34 %	91.47 %	92.95 %	90.97 %	92.58 %	92.83 %	92.70 %
240	92.33 %	92.33 %	93.07 %	91.84 %	93.07 %	92.45 %	92.08 %
250	91.47 %	91.84 %	92.45 %	91.22 %	92.33 %	92.95 %	92.45 %
<b>Máximo</b>	<b>92.33 %</b>	<b>92.33 %</b>	<b>93.07 %</b>	<b>92.21 %</b>	<b>93.07 %</b>	<b>92.95 %</b>	<b>93.44 %</b>

## Extracción de características

La imagen ingresa por tres diferentes secciones de extracción de características, cada una extrae diferentes características utilizando un kernel de diferente tamaño. La

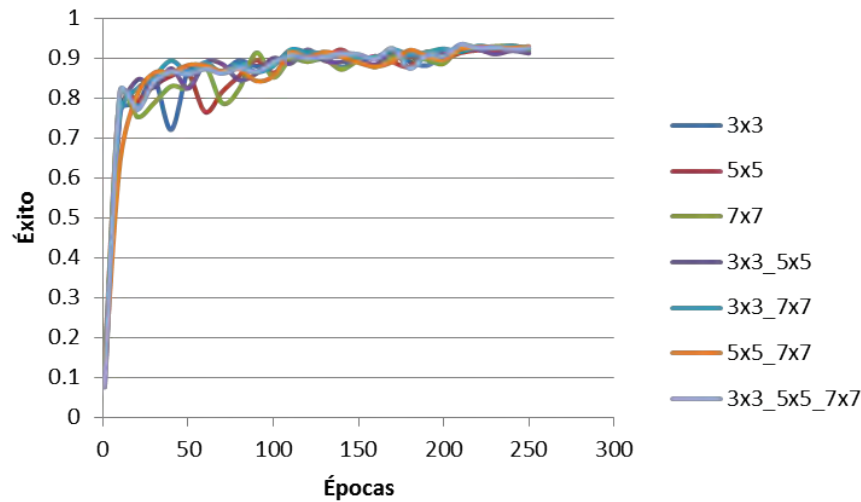


**Figura 3.9.** Grafo de la arquitectura propuesta.

primera sección utiliza un kernel de  $3 \times 3$  en sus cinco capas. La segunda sección utiliza en la primera capa un kernel de  $5 \times 5$  y en las siguientes cuatro se cambian a  $3 \times 3$ . En la tercera sección, la primera capa utiliza un kernel de  $7 \times 7$ , en el segundo uno de  $5 \times 5$  y en las últimas tres es de  $3 \times 3$ . Las primeras cuatro capas de cada sección de extracción utilizan convolución, normalización, ReLU y max-pooling. En la última capa de cada sección utilizan convolución, normalización, ReLU y avg-pooling. A continuación se explica de manera más detallada el funcionamiento de cada una de las secciones de extracción de características:

- **Imagen de entrada:** la arquitectura inicia con la capa de entrada que hace un pre-procesamiento de la imagen para que tenga como dimensiones  $224 \times 224$  y una profundidad de tres ya que la imagen ingresa con el formato RGB.
- **Sección de extracción de características uno:**





**Figura 3.10.** Gráfica con las diferentes pruebas de la red con el conjunto de imágenes de Oliva & Torralba.

- **Primera capa:**

1. El grupo de capas inicia con la operación de convolución entre la matriz de  $224 \times 224$  de la capa anterior y un núcleo propuesto de  $3 \times 3$ , con esto se obtienen características de bajo nivel (es decir, bordes, líneas y curvas). Se realizará con una profundidad propuesta de 32 (es el número de imágenes a obtener con filtros diferentes).
2. Se realiza una normalización con los resultados de la capa anterior. La capa de normalización propone añadir la capacidad de generalización de la red.
3. Continúa con una operación de unidad lineal rectificada «Rectified Linear Units, ReLU». Las unidades ReLU generan pocas representaciones con cero que parecen ajustarse a los datos de las imágenes digitales. Las unidades ReLU alcanzan un buen rendimiento, aunque su entrenamiento requiere de grandes cantidades de imágenes ya clasificadas.
4. Se realiza un max-pooling disminuyendo la imagen a la mitad, lo que significa una reducción a  $112 \times 112$  continuando con la misma profundidad.

- **Segunda capa:**
  5. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 48 filtros.
  6. Se realiza una normalización con los resultados de la capa anterior.
  7. Continúa con una operación ReLU.
  8. Se realiza un max-pooling, lo que significa una reducción a  $56 \times 56$ .
- **Tercera capa:**
  9. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 64 filtros.
  10. Se realiza una normalización con los resultados de la capa anterior.
  11. Continúa con una operación ReLU.
  12. Se realiza un max-pooling, lo que significa una reducción a  $28 \times 28$ .
- **Cuarta capa:**
  13. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 80 filtros.
  14. Se realiza una normalización con los resultados de la capa anterior.
  15. Continúa con una operación ReLU.
  16. Se realiza un max-pooling, lo que significa una reducción a  $14 \times 14$ .
- **Quinta capa:**
  17. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 96 filtros.
  18. Se realiza una normalización con los resultados de la capa anterior.
  19. Continúa con una operación ReLU.
  20. Se realiza un avg-pooling, lo que significa una reducción a  $8 \times 8$ .
- **Sección de extracción de características dos:**
  - **Primera capa:**
    1. Inicia con la operación de convolución entre la matriz de  $224 \times 224$  de la imagen de entrada y un núcleo de  $5 \times 5$  con una profundidad de 32.
    2. Se realiza una normalización con los resultados de la capa anterior.

3. Continúa con una operación ReLU.
  4. Se realiza un max-pooling, disminuyendo la imagen a  $112 \times 112$  continuando con la misma profundidad.
- **Segunda capa:**
    5. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 48 filtros.
    6. Se realiza una normalización con los resultados de la capa anterior.
    7. Continúa con una operación ReLU.
    8. Se realiza un max-pooling, lo que significa una reducción a  $56 \times 56$ .
  - **Tercera capa:**
    9. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 64 filtros.
    10. Se realiza una normalización con los resultados de la capa anterior.
    11. Continúa con una operación ReLU.
    12. Se realiza un max-pooling, lo que significa una reducción a  $28 \times 28$ .
  - **Cuarta capa:**
    13. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 80 filtros.
    14. Se realiza una normalización con los resultados de la capa anterior.
    15. Continúa con una operación ReLU.
    16. Se realiza un max-pooling, lo que significa una reducción a  $14 \times 14$ .
  - **Quinta capa:**
    17. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 96 filtros.
    18. Se realiza una normalización con los resultados de la capa anterior.
    19. Continúa con una operación ReLU.
    20. Se realiza un avg-pooling, lo que significa una reducción a  $8 \times 8$ .
- **Sección de extracción de características tres:**
    - **Primera capa:**
      1. Inicia con la operación de convolución entre la matriz de  $224 \times 224$

de la imagen de entrada y un núcleo de  $7 \times 7$ , con una profundidad de 32.

2. Se realiza una normalización con los resultados de la capa anterior.
3. Continúa con una operación ReLU.
4. Se realiza un max-pooling, disminuyendo la imagen a  $112 \times 112$  continuando con la misma profundidad.

- **Segunda capa:**

5. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $5 \times 5$ . Aumentando a 48 filtros.
6. Se realiza una normalización con los resultados de la capa anterior.
7. Continúa con una operación ReLU.
8. Se realiza un max-pooling, lo que significa una reducción a  $56 \times 56$ .

- **Tercera capa:**

9. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 64 filtros.
10. Se realiza una normalización con los resultados de la capa anterior.
11. Continúa con una operación ReLU.
12. Se realiza un max-pooling, lo que significa una reducción a  $28 \times 28$ .

- **Cuarta capa:**

13. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 80 filtros.
14. Se realiza una normalización con los resultados de la capa anterior.
15. Continúa con una operación ReLU.
16. Se realiza un max-pooling, lo que significa una reducción a  $14 \times 14$ .

- **Quinta capa:**

17. Continúa con la operación de convolución entre la última matriz de la capa anterior y un núcleo de  $3 \times 3$ . Aumentando a 96 filtros.
18. Se realiza una normalización con los resultados de la capa anterior.
19. Continúa con una operación ReLU.
20. Se realiza un avg-pooling, lo que significa una reducción a  $8 \times 8$ .

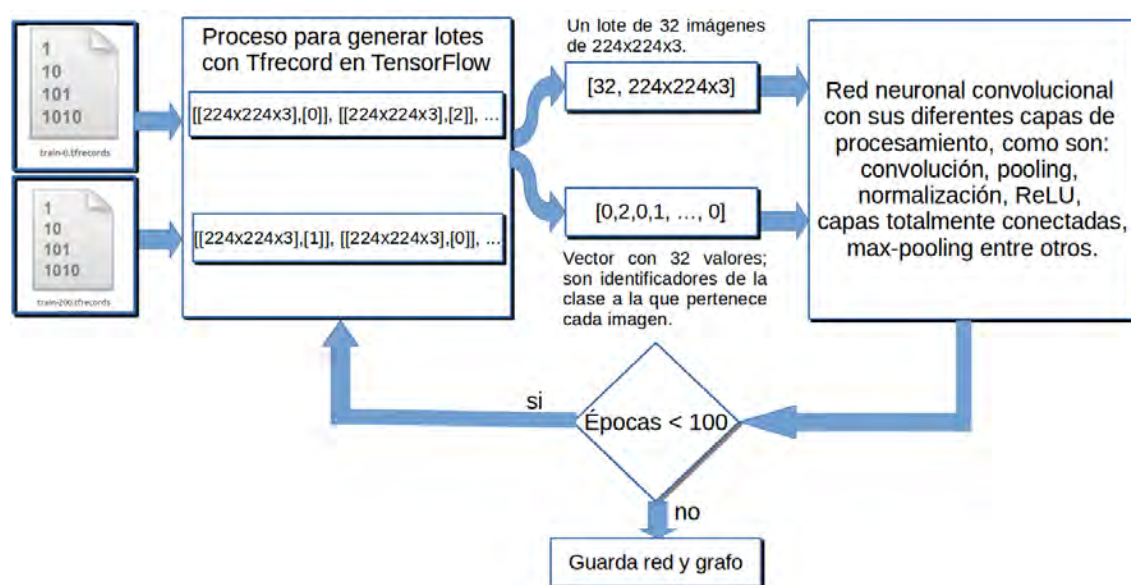
## Clasificación

La salida de cada sección de extracción de características se concatena para generar un vector unidimensional. Así se puede continuar con las capas totalmente conectadas para realizar la clasificación, como se describe a continuación:

- **Concatenación de las salidas de las tres secciones:** la clasificación inicia con la concatenación de la salida de cada red, se tendrá un vector con dimensiones  $18432 \times 1$  (características de la imagen). Notando que el resultado de cada uno es un escalar.
- **Primera capa totalmente conectada:**
  - 21. Se realizará la primera capa totalmente conectada entre el vector concatenado anteriormente y una nueva con una profundidad de 2048.
  - 22. Continúa con una capa de ReLU.
  - 23. Después con la capa de abandono «dropout».
- **Segunda capa totalmente conectada:**
  - 24. De los 2048 nodos de la capa anterior se interconectan con 1024 nodos.
  - 25. Continúa con una capa de ReLU.
  - 26. Continúa con la capa de dropout.
- **Tercera capa totalmente conectada:**
  - 27. El grupo inicia con la capa totalmente conectada con los nodos de la capa anterior y el número de categorías (NC) que tiene el conjunto de imágenes de entrenamiento. Es importante hacer notar que si se cambia de conjunto de imágenes se tendrá que modificar el número de categorías.
  - 28. Softmax es una función de regresión que nos ayuda a clasificar múltiples categorías ingresadas en la base de datos de entrenamiento.
- **Capa de salida:** al final, en la última capa nos mostrará el porcentaje de éxito de que la imagen digital que se ingresó sea un perro, un gato, etc.

### 3.5. Entrenamiento de la red

Una vez generados los archivos de entrenamiento, éstos se deben cargar en la red pero debe existir un módulo que decodifique los datos para procesarlos como imágenes de entrada, en la figura 3.11 se muestra el proceso. Primero se leen los archivos tipo tfrecord, después se generan lotes de imágenes y lotes de etiquetas (describen a que clase pertenece cada imagen). Luego se ingresa el lote a la red neuronal convolucional, empieza a iterar hasta cumplir el número de épocas programado, en este proceso los pesos cambian en cada iteración y con diferentes lotes, al final guarda la red entrenada y el grafo, esto para poder realizar las pruebas más adelante.



**Figura 3.11.** Proceso para cargar archivos tfrecord en TensorFlow y generar lotes de imágenes.

En la figura 3.11 se muestra que se necesitan principalmente dos parámetros de entrada a la red. La primera variable es la que recibe la imagen o grupo de imágenes; esta variable se puede representar como un vector multidimensional de nombre "x". La segunda variable, es la clase a la que pertenece la imagen o imágenes; esta variable se puede representar con el nombre de "y" de tamaño  $[32, \text{número de clases}]$ . La variable "x" es de tamaño  $[32, 224, 224, 3]$ , está debe tener el mismo tamaño de la imagen de entrada ( $224 \times 224 \times 3$ ) y el tamaño del lote (32 en nuestro

caso).

La red propuesta es entrenada inicialmente con los diferentes conjuntos de datos de entrenamiento utilizando los siguientes parámetros: algoritmo de entrenamiento ADAM [74], con un tamaño de lote de  $\beta = 32$  imágenes y un decaimiento de pesos (factor de regularización) de  $\lambda = 0.00004$ . Los pesos iniciales en cada una de las capas fueron inicializados con una distribución gaussiana con una media de 0 y una desviación estándar de 0.01. Los umbrales de activación en cada una de las capas fueron inicializados a cero. Iniciamos con una tasa de aprendizaje de  $\alpha = 0.001$ , la cual se disminuyó en un factor de 10 después de cada 25 épocas para tener cambios de aprendizaje más específicos en 100 épocas de entrenamiento. AlexNet, GoogleNet y ResNet fueron entrenadas con los mismos parámetros para su comparación con la red propuesta.

Para iniciar el entrenamiento de cualquier red se lee el tamaño de la imagen, el total de imágenes, número de clases, entre otros parámetros importantes guardados al momento de generar los archivos tfrecord, estos se describen en los siguientes puntos.

Carpetas de lectura y escritura de datos:

- Lugar en el que se guardará el modelo generado por el entrenamiento así como el grafo.
- Lugar en dónde se encuentran los archivos tfrecord.

Parámetros iniciales:

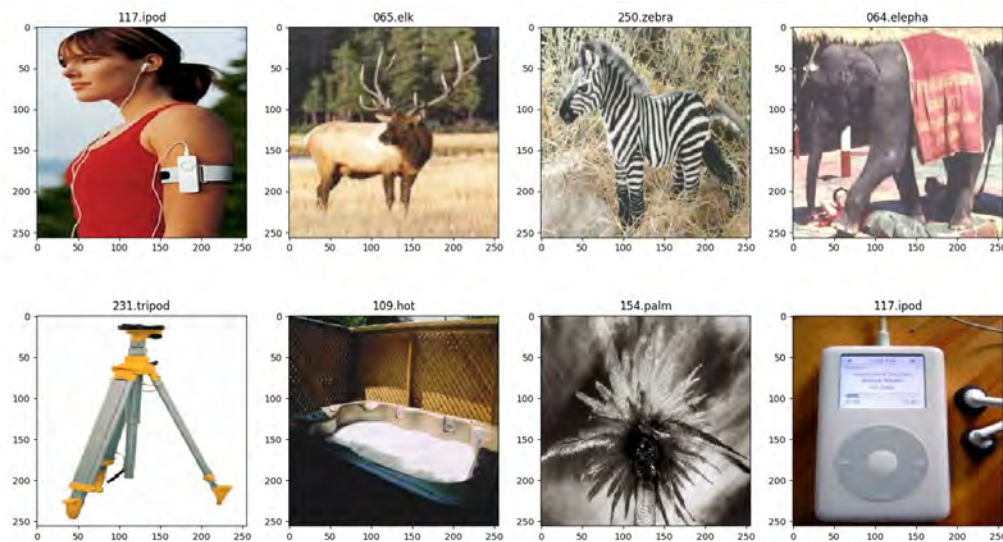
- Tasa de aprendizaje, este va a cambiar dependiendo del número de épocas.
- Número de épocas.
- Tamaño de lote.
- Número de canales, para RGB es de 3.

Del archivo generado por TFpyToolbox de nombre "Total.txt" se leen los siguientes parámetros:

- Número de clases.
- Tamaño de la imagen.
- Número de imágenes por dataset.

- Número de imágenes del total de la base de datos.
- Imágenes por archivo tfrecord.

De los archivos tfrecord se leen las imágenes y las clases separándolos por lotes y de manera aleatoria, TensorFlow tiene la función `shuffle_batch` para hacerlo, esta función regresa dos lotes, uno de imágenes y otro de clases. Cuando se obtienen las imágenes se pueden visualizar algunas para verificar que están de forma correcta y poder continuar con el proceso como se muestra en la figura 3.12.



**Figura 3.12.** Lectura de imágenes del dataset.

## 3.6. Prueba de la red

Para realizar las pruebas se carga la red entrenada y se evalúa con el conjunto de imágenes de prueba (los archivos fueron generados por el Toolbox descrito en el apéndice B). En la figura 3.13 se describe el proceso para realizar las pruebas. Se genera un nuevo código especial para las pruebas donde se carga el grafo con los pesos pero se quita el método de aprendizaje (ya no será necesario), se modifica el dropout entre otros parámetros. También en esta parte ya no es necesario extraer las imágenes de manera aleatoria, se tendrá que modificar el código de extracción de imágenes agregando `tf.train.batch` en vez de `tf.train.shuffle_batch`.



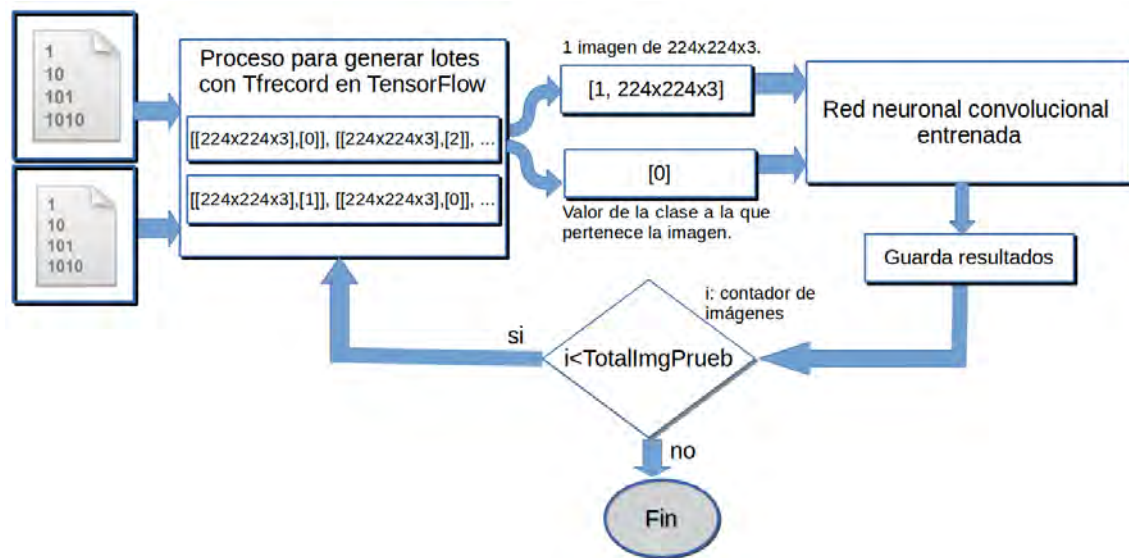


Figura 3.13. Proceso para realizar las pruebas.

## Experimentos realizados a la red

Para evaluar el rendimiento y hacer la comparación con otras redes se realizaron fundamentalmente dos pruebas:

1. Evaluación del rendimiento de la red en la cual se utilizan los conjuntos de datos de entrenamiento. Las imágenes fueron redimensionadas a  $(224 \times 224)$  y se ingresó directamente a las distintas secciones. En esta prueba se utilizó una GPU Titan X con 12 GB de memoria RAM.
2. Se utilizaron los mismos conjuntos de imágenes pero esta vez fueron redimensionadas a  $256 \times 256$ , después se hizo un recorte de manera aleatoria de tamaño  $224 \times 224$ . Después se modificó el brillo, el contraste y se giró la imagen de manera aleatoria. Finalmente, se restó la media y se dividió por la varianza de los píxeles. Con esto provocamos un aumento de datos para simular más imágenes de las que hay. Al realizar los pre-procesamientos se ha tenido que aumentar las épocas a 250, por lo tanto aumentó el tiempo de entrenamiento. Se tuvo que utilizar cuatro GPUs NVIDIA GEFORCE GTX 1080 para reducir el tiempo de entrenamiento entrenando tres veces más rápido, dejando un lote 32 imágenes por cada GPU.

## Análisis de datos para evitar problemas de memoria en GPU

Para el entrenamiento de la red se debe tomar en cuenta el hardware. En este caso utilizamos primero una GPU con 12 Gb de memoria RAM. No se puede utilizar toda la memoria para el procesamiento, ya que el sistema operativo utilizado (Ubuntu 16.04) requiere por lo menos 512 Mb para su correcto funcionamiento. También debemos dejar espacio libre para no saturar la memoria, nos queda mas o menos 11 Gb. Tomando en cuenta esto podemos hacer el cálculo con base en la cantidad de información que vamos a procesar. Primero, podemos ver en el grafo 3.9 la cantidad de datos que tendrán que soportar los tensores en TensorFlow. Si se ingresa una imagen de  $224 \times 224 \times 3$  se tendrá que multiplicar por el número de lotes y por la cantidad de bytes utilizados, si es de punto flotante será de 4; por lo tanto, el número de tensores utilizados en el grafo sería  $224 \times 224 \times 3 \times 32 \times 4 \times 1$ . El total final sería de 19,267,584 bytes, dividiendo entre  $1024 \times 1024$  nos dará 18.375 Mega bytes a utilizar. Así se podría calcular el tamaño de datos sin sobrepasar la memoria de la GPU.

# Capítulo 4

## Resultados experimentales

En este capítulo se describen los resultados de los dos experimentos realizados. Después se muestra un análisis del proceso de entrenamiento y prueba de nuestra red.

Para evaluar el rendimiento de las redes se entrenó con cada conjunto de imágenes; éstas fueron divididas en entrenamiento y prueba. Los conjuntos de entrenamiento fueron formados con el 70 % de las imágenes de cada base de datos y el 30 % restante se utilizó para formar los conjuntos de prueba; la selección de las imágenes se realizó de manera aleatoria. Para ajustar las redes a cada uno de los conjuntos de entrenamiento fue necesario igualar el número de neuronas de salida al número de clases de cada conjunto. El nombre que se le dio a nuestra red es ToniNet para poder diferenciarla de las otras redes.

Se realizaron dos tipos de pruebas, la primera fue sin aumento de datos y la segunda con aumento de datos, como se describe en las siguientes secciones.

### 4.1. Primera prueba sin aumento de datos

En la primera prueba se ingresa directamente la imagen a cada sección de extracción de características de la red. Para realizar el entrenamiento se utilizó una GPU Titan X. El entrenamiento se hizo con 100 épocas donde las redes aprendieron casi en un 100 % las imágenes de entrenamiento, pero al realizar las pruebas bajó considerablemente. Los resultados de la prueba se muestran en la tabla 4.1, describiremos cada conjunto de imágenes.

En la columna del conjunto de imágenes de Oliva & Torralba se puede apreciar

## 4. Resultados experimentales

---

**Tabla 4.1.** Resultados sin aumento de datos con 100 épocas y una GPU.

CNN	Oliva & Torralba			Stanford Dogs			Caltech 256		
	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5
AlexNet	23	86.5 %	99.5 %	122	33.4 %	64.4 %	147	49.9 %	70.1 %
GoogleNet	27	88.0 %	100 %	175	30.5 %	64.5 %	215	50.8 %	71.8 %
ResNet 152	106	85.5 %	99.5 %	760	14.7 %	41.0 %	890	32.9 %	54.5 %
ToniNet	47	91.5 %	100 %	419	39.7 %	68.8 %	615	52.7 %	72.0 %

que nuestra red es la única que supera el 90 % a comparación de las otras tres redes. Este conjunto de imágenes es el que tiene menor cantidad de imágenes y menor cantidad de clases (8 en total). El tiempo de entrenamiento de nuestra red es mayor que AlexNet y GoogleNet pero es más rápido que ResNet 152 y con mejor rendimiento.

El conjunto de imágenes de Stanford Dogs tiene 120 clases de diferentes razas de perros. En la columna que le corresponde se puede observar que nuestra red supera en más del 6 % al más cercano que es AlexNet. También se puede ver que ResNet 152 es el que tiene menor rendimiento con 14.7 %, esto se puede explicar ya que tiene más capas a comparación de las otras redes, por lo tanto necesita más imágenes para que pueda obtener mejor rendimiento.

El conjunto de imágenes Caltech 256 tiene 256 clases de diferentes objetos, también incluye una clase extra con imágenes de diferentes escenas. Nuestra red tiene mejor rendimiento que las otras redes pero el porcentaje es menor. El rango de imágenes entre las clases varía entre 80 a 827, aquí hay un problema ya que al entrenar, la red aprende características de las clases con mayor cantidad de imágenes.

Nos dimos a la tarea de investigar algún método para mejorar el rendimiento para bases de datos con pocas imágenes, el aumento de datos fue el que seleccionamos por su amplio uso y se explica en la siguiente prueba.

### 4.2. Segunda prueba con aumento de datos

En la segunda prueba utilizamos el aumento de datos, esto para tratar de mejorar el rendimiento que se obtuvo en la primera prueba. El aumento de datos nos ayuda a simular que existen más imágenes de las que en realidad tenemos. Se utilizó el mismo conjunto de imágenes de la primera prueba pero, esta vez, redimensionadas

**Tabla 4.2.** Resultados con aumento de datos y con 250 épocas, con cuatro GPUs.

CNN	Oliva & Torralba			Stanford Dogs			Caltech 256		
	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5
AlexNet	19	90.4 %	99.8 %	175	43.2 %	70.8 %	213	51.8 %	70.6 %
GoogleNet	14	91.5 %	99.8 %	164	51.9 %	80.2 %	196	60.6 %	79.0 %
ResNet 152	85	92.8 %	99.8 %	538	53.6 %	82.7 %	739	64.7 %	81.8 %
ToniNet	34	94.6 %	100 %	255	55.2 %	84.7 %	371	62.5 %	80.8 %

a  $256 \times 256$ . También se realizó un preprocesamiento al momento de ingresar la imagen a la red: se le hizo un recorte de manera aleatoria al tamaño de  $224 \times 224$  para que la red pueda aceptarla, se modificó el brillo, el contraste y se giró la imagen de manera aleatoria. Al realizar el entrenamiento ya no fue suficiente utilizar 100 épocas así que se incrementó a 250 épocas. Se incrementó la cantidad de operaciones lo que implicó aumento en el tiempo de procesamiento de datos, así que se decidió aumentar el número de GPUs. Utilizamos 4 GPUs GTX 1080 para disminuir el tiempo de entrenamiento. Los resultados de las pruebas con aumento de datos se muestran en la tabla 4.2.

Al revisar los resultados del conjunto de imágenes de Oliva & Torralba se puede apreciar que se mejoró el rendimiento de todas las redes. Esta vez todas las redes lograron superar el 90 % de éxito en top 1. Nuestra red obtuvo un mejor rendimiento superando al más cercano en 1.8 % en top 1 y llegando a un 100 % en top 5. ResNet 152 se puede destacar ya que mejoró quedando en segundo lugar con 92.8 %.

Con el conjunto de imágenes de Stanford Dogs nuestra red mejoró en más del 14 % comparándola con la primera prueba, logrando un 55.2 % de éxito; las otras redes también mejoraron destacando nuevamente a ResNet con una mejora de 38.9 %, logrando un 53.3 % pero con mayor cantidad de tiempo en entrenamiento. GoogleNet es el que necesita menos tiempo para cumplir la cantidad de épocas con 164 minutos. AlexNet es el que tiene menor rendimiento con 43 %.

Al realizar la prueba con Caltech 256 nuestra red quedó en segundo lugar con 62.5 %, siendo superada por ResNet con 64.7 %.

Utilizando los 4 GPUs se logra reducir los tiempos de entrenamiento. Por ejemplo, ResNet se tardó en entrenar a Caltech 256 aproximadamente 739 minutos utilizando los 4 GPUs con 250 épocas y con aumento de datos lo que implica mayor procesamiento, mientras que con una GPU se tardó 890 minutos con 100 épocas.

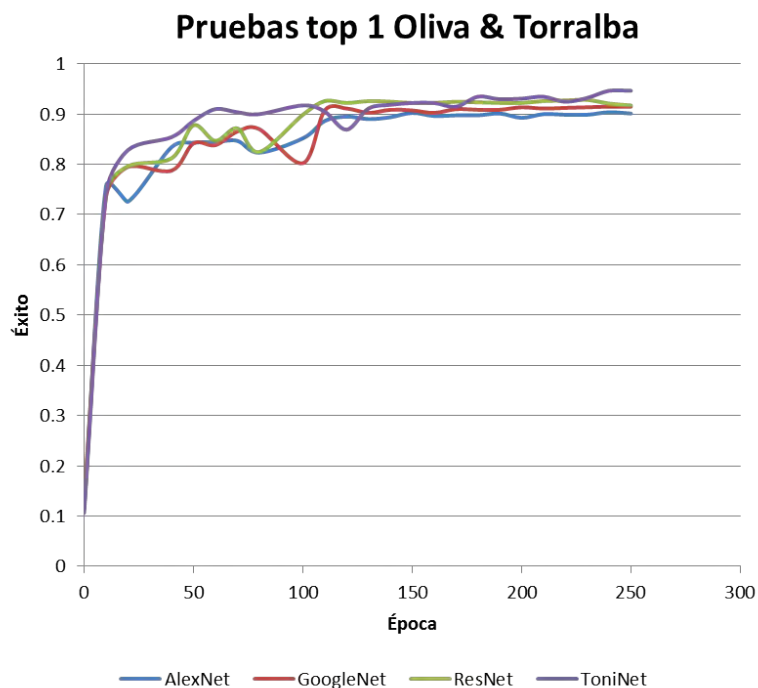
Se decidió aumentar la cantidad de épocas a 500 para ver la mejora del rendimiento, lo que implica mayor tiempo de entrenamiento. Sin embargo el rendimiento no incrementó en más del 2%.

### 4.2.1. Rendimiento de redes por cada conjunto de imágenes

En las siguientes secciones se puede ver de forma gráfica como fue incrementándose el rendimiento al paso de las épocas, haciendo una comparación entre las redes por cada conjunto de imágenes.

#### Pruebas con Oliva & Torralba

Al realizar las evaluaciones por cada época (incluyendo cuando la red aun no se entrena), podemos analizar el comportamiento del aprendizaje al paso de las épocas en pruebas con top 1 como se muestra en la figura 4.1.

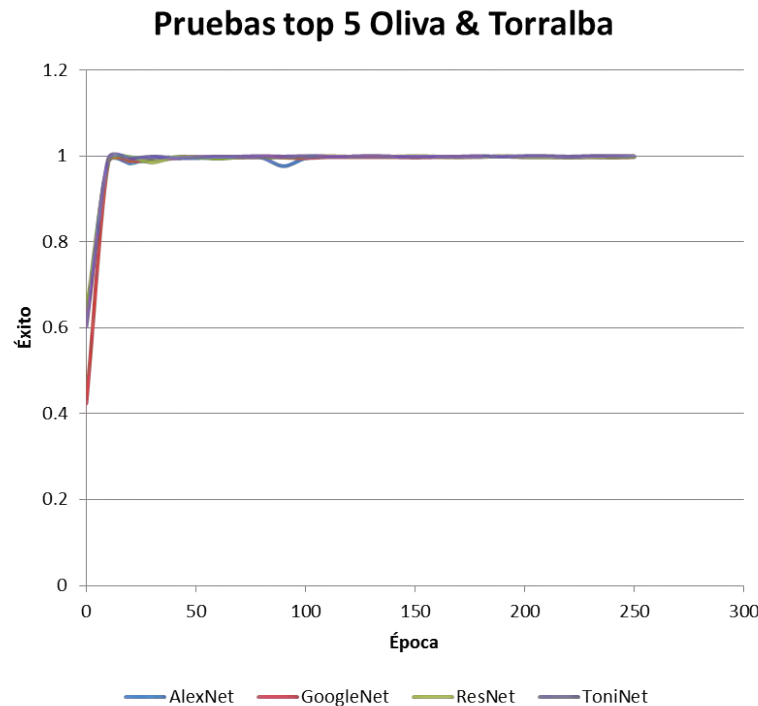


**Figura 4.1.** Rendimiento de las redes en pruebas en top 1 con el conjunto de datos Oliva & Torralba.

Las redes van mejorando a medida que pasan las épocas pero llega el momento en que deja de hacerlo, aproximadamente en la época 130. Nuestra red pareciera que

seguirá mejorando pero al seguir entrenando hasta llegar a 500 épocas el incremento no fue mayor al 1%. Se puede destacar de ToniNet que el aprendizaje es más constante y supera a las otras redes en rendimiento.

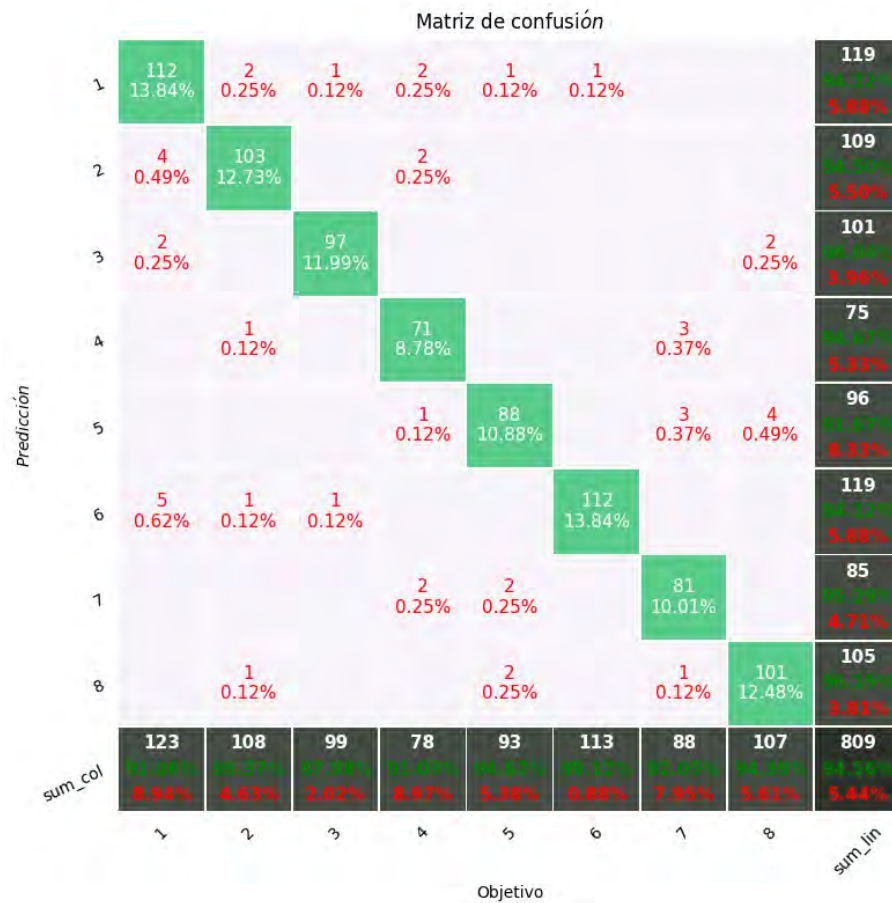
En la figura 4.2 podemos ver el rendimiento en top 5 de las cuatro redes. El aprendizaje de las redes llega casi a un 100 % desde la época 10, se puede explicar ya que solo son 8 clases.



**Figura 4.2.** Rendimiento de las redes en pruebas en top 5 con el conjunto de datos Oliva & Torralba.

Como complemento, y para mejor análisis de datos, para visualizar las clases que más se confunden se generaron matrices de confusión por cada prueba como se muestra en la figura 4.3. Se puede ver que en la diagonal se obtiene un 94.56 % de éxito. Con 765 imágenes correctas de 809. De forma horizontal se puede ver que la clase con mayor éxito en aprendizaje es *Tallbuilding* con 96.19 % seguida de *Forest* con 96.04 %. La clase con menor éxito es *Inside\_city* con 91.67 %, confundiéndose con *Street* y *Tallbuilding*. De forma vertical *Highway* es la más confundida con *Opencountry*, *Coast* y *Street*, también *Opencountry* es confundida con *Coast*, *Forest* y *Mountain*. *Mountain* es la clase que prácticamente no es confundida.

## 4. Resultados experimentales



**Figura 4.3.** Matriz de confusión con el conjunto de datos Oliva & Torralba, con las clases  $C1, C2, \dots, C8$  que corresponden a *Opencountry*, *Coast*, *Forest*, *Highway*, *Inside\_city*, *Mountain*, *Street* y *Tallbuilding*; tiene un éxito en pruebas de 94.56 % con 765 imágenes correctas de 809. Resultados de la evaluación de la red entrenada ToniNet.

En la figura 4.4 se muestra el resultado en pruebas. Se puede ver que la red ToniNet confundió una imagen de la clase *costa* por una de la clase *montaña*.

### Pruebas con Stanford Dogs

En la figura 4.5 se puede ver el rendimiento de cada red con el conjunto de imágenes de Stanford Dogs. Se muestra que ToniNet tiene mejor rendimiento que las otras redes, obteniendo un 55.2 % de éxito en 250 épocas. ToniNet se entrenó hasta llegar a 500 épocas logrando un 56.2 % como máximo en rendimiento. El rendimiento es poco significativo comparado al tiempo de procesamiento de datos. Después de la época 50 el aprendizaje es más lento para todas las redes. El que tuvo menor rendimiento





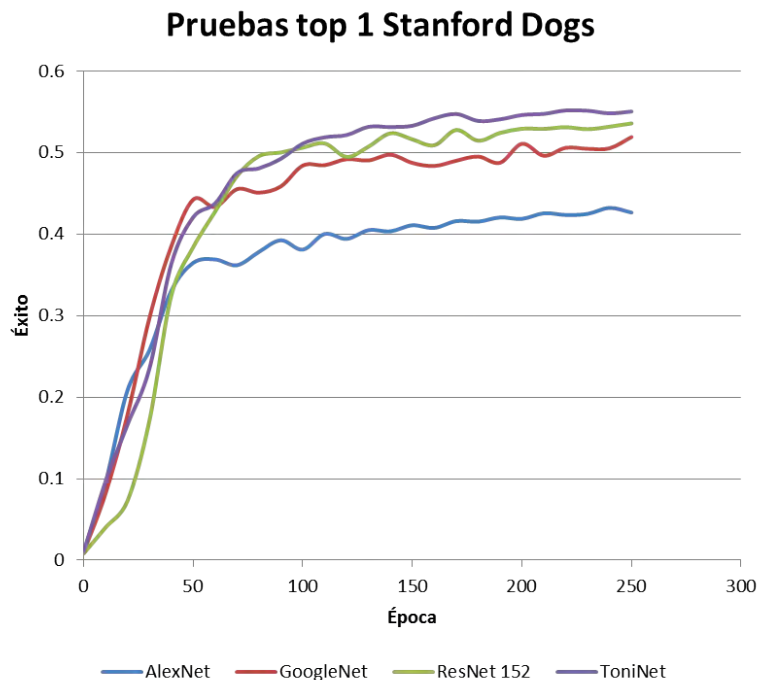
**Figura 4.4.** Prueba con Oliva & Torralba, la letra *R* significa que es la clasificación real, la letra *P* significa que es la clasificación dada por la red.

fue AlexNet.

La evaluación de las redes en top 5 se muestra en la figura 4.6. Como en la primera prueba, ToniNet tiene mejor rendimiento pero se puede ver que el aprendizaje tiende a mejorar al paso de las épocas, aunque es poco significativo.

### Pruebas con Caltech 256

Con este conjunto de datos nuestra red obtuvo un rendimiento de 62.5% y 80.8% respectivamente en top 1 y top 5 hasta la época 250. ResNet 152 fue la que obtuvo un mejor rendimiento con 64.7% en top 1 y 81.8% en top 5. En la figura 4.7 se muestra el comportamiento de las redes en pruebas con top 1. Hasta la época 70 se



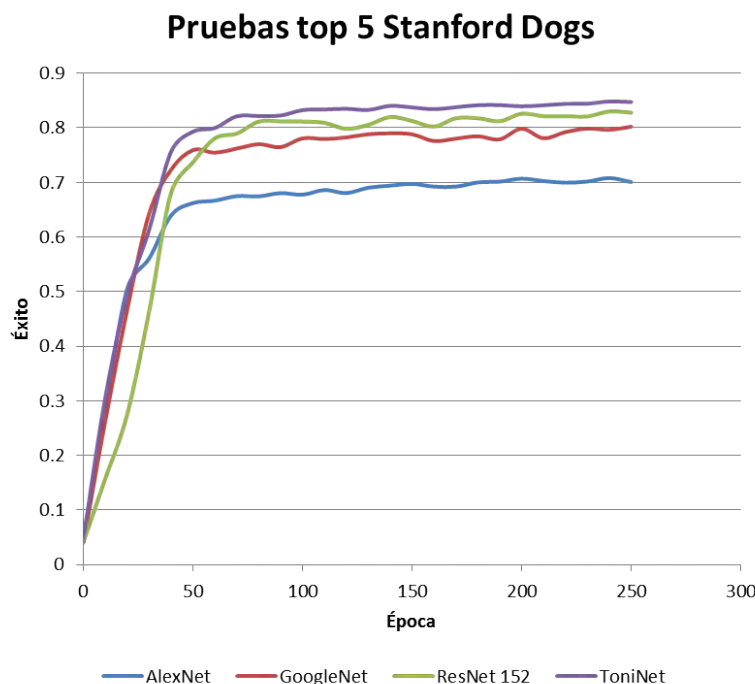
**Figura 4.5.** Rendimiento de las redes en pruebas en top 1 con el conjunto de imágenes Stanford Dogs.

puede ver que las redes aprenden pero después el incremento en rendimiento ya no es significativo, a partir de la época 250 el aprendizaje se mantiene constante. En la figura 4.8 se muestra el rendimiento de las redes en top 5, donde nuestra red casi está a la par de ResNet, por otro lado AlexNet es el que tiene menor rendimiento.

### 4.3. Resultados del entrenamiento de la red ToniNet

Cuando la red es entrenada normalmente se tiene un rendimiento alto, por ejemplo con Oliva & Torralba llega a un 99% en top 1. Los resultados varían dependiendo del conjunto de imágenes. Al utilizar aumento de datos decrece el rendimiento en el entrenamiento, pero da mejores resultados en las pruebas.

El error de pérdida es importante analizarlo al momento de estar entrenando, ya que al ir disminuyendo aumenta el rendimiento, pero si se mantiene constante la red ya no aprende, así podríamos evitar estar entrenando por varias épocas. En la

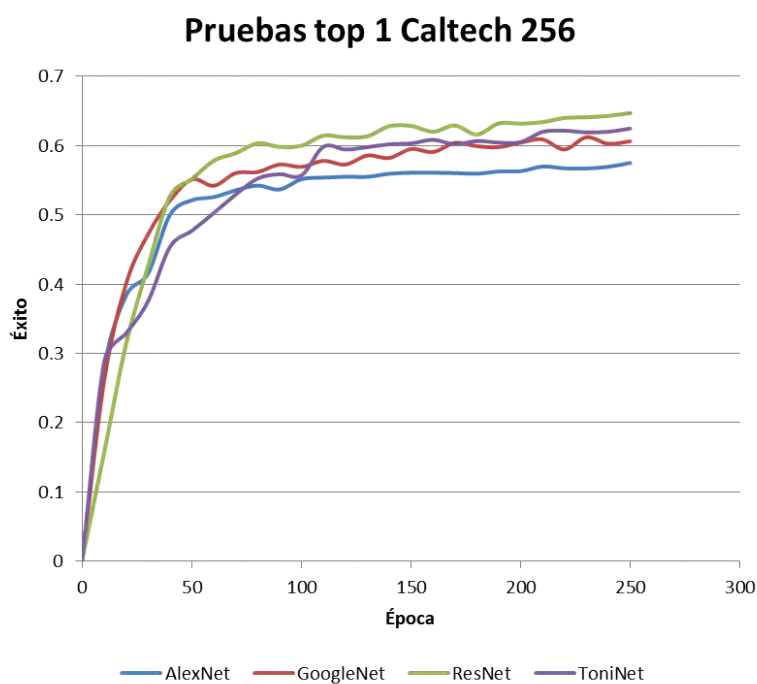


**Figura 4.6.** Rendimiento de las redes en pruebas en top 5 con el conjunto de imágenes Stanford Dogs.

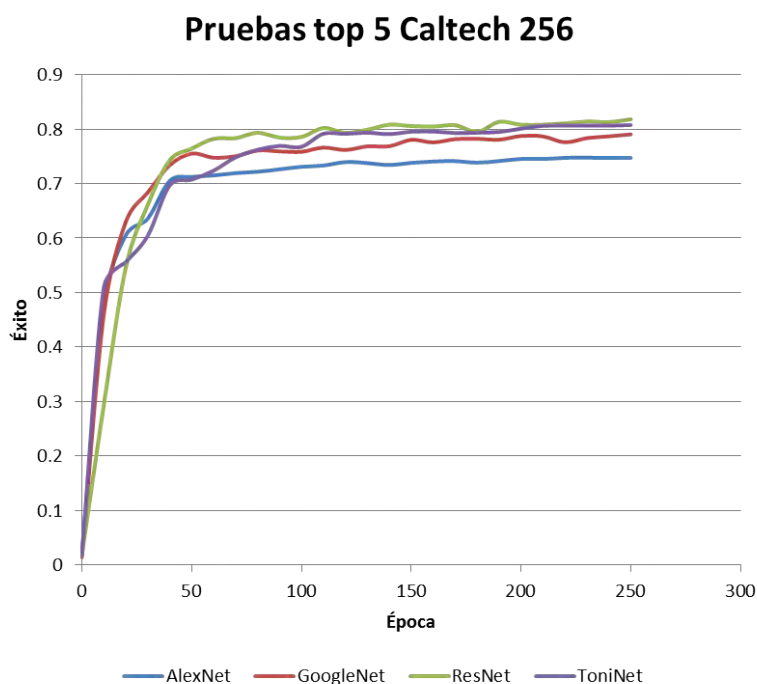
figura 4.9 se muestra el error de pérdida para cada conjunto de imágenes. Se puede ver que al entrenar con el conjunto de imágenes de Oliva & Torralba se llega a casi el 0 % del error antes de la época 100. Por otro lado, para entrenar los otros conjuntos de imágenes se tardan más tiempo.

Los resultados del entrenamiento en top 1 se puede ver en la figura 4.10. En la imagen se muestra que Oliva & Torralba aprende en casi el 100 % las imágenes de entrenamiento, con los otros dos conjuntos de imágenes llega a un 98 % como máximo y se tardan más tiempo en realizar el aprendizaje. Los resultados del entrenamiento en top 5 se puede ver en la figura 4.11. En la figura se puede ver que llega a un 100 % en el aprendizaje de todos los conjuntos de imágenes.

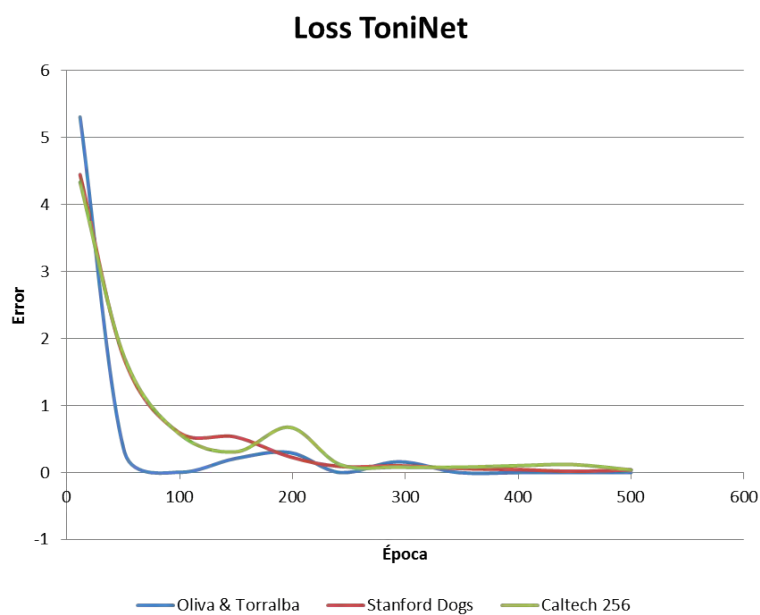
Finalmente se puede visualizar las imágenes con la clasificación real y la que clasificación que da la red con las imágenes de entrenamiento, como se muestra en la figura 4.12 y 4.13, esto para validar los resultados del entrenamiento de forma visual. También se puede ver en la imagen 4.14 las clasificaciones de la red, con el conjunto de imágenes Stanford Dogs de prueba.



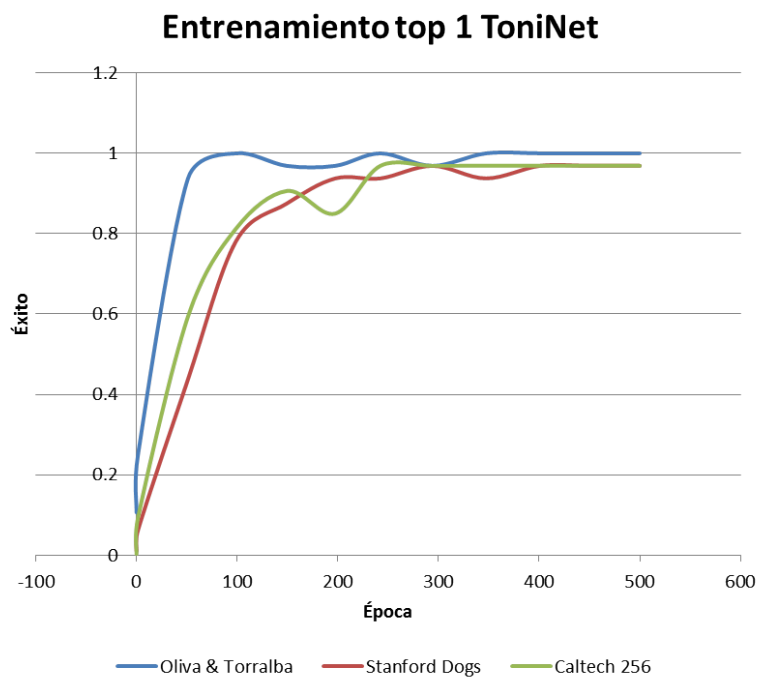
**Figura 4.7.** Rendimiento de las redes en pruebas en top 1 con el conjunto de imágenes Caltech 256.



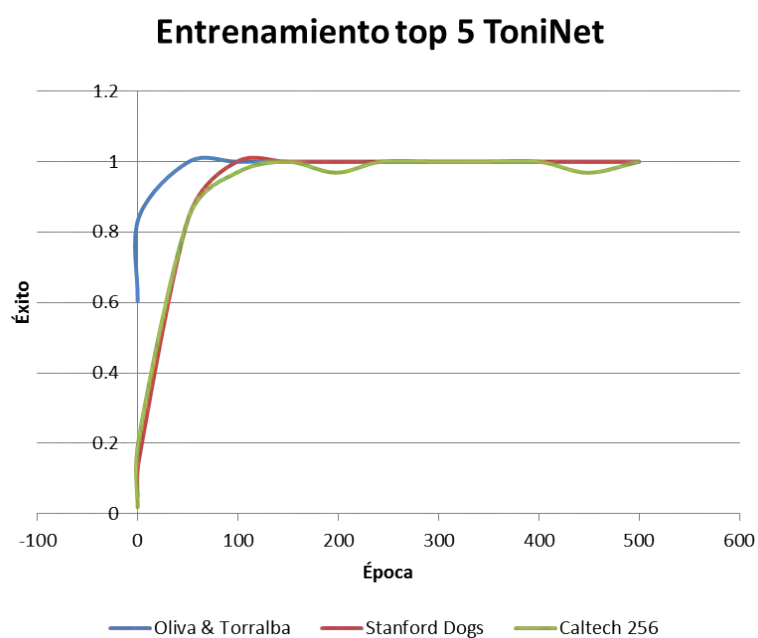
**Figura 4.8.** Rendimiento de las redes en pruebas en top 5 con el conjunto de imágenes Caltech 256.



**Figura 4.9.** Disminución del error con diferentes conjuntos de imágenes usando la red ToniNet.

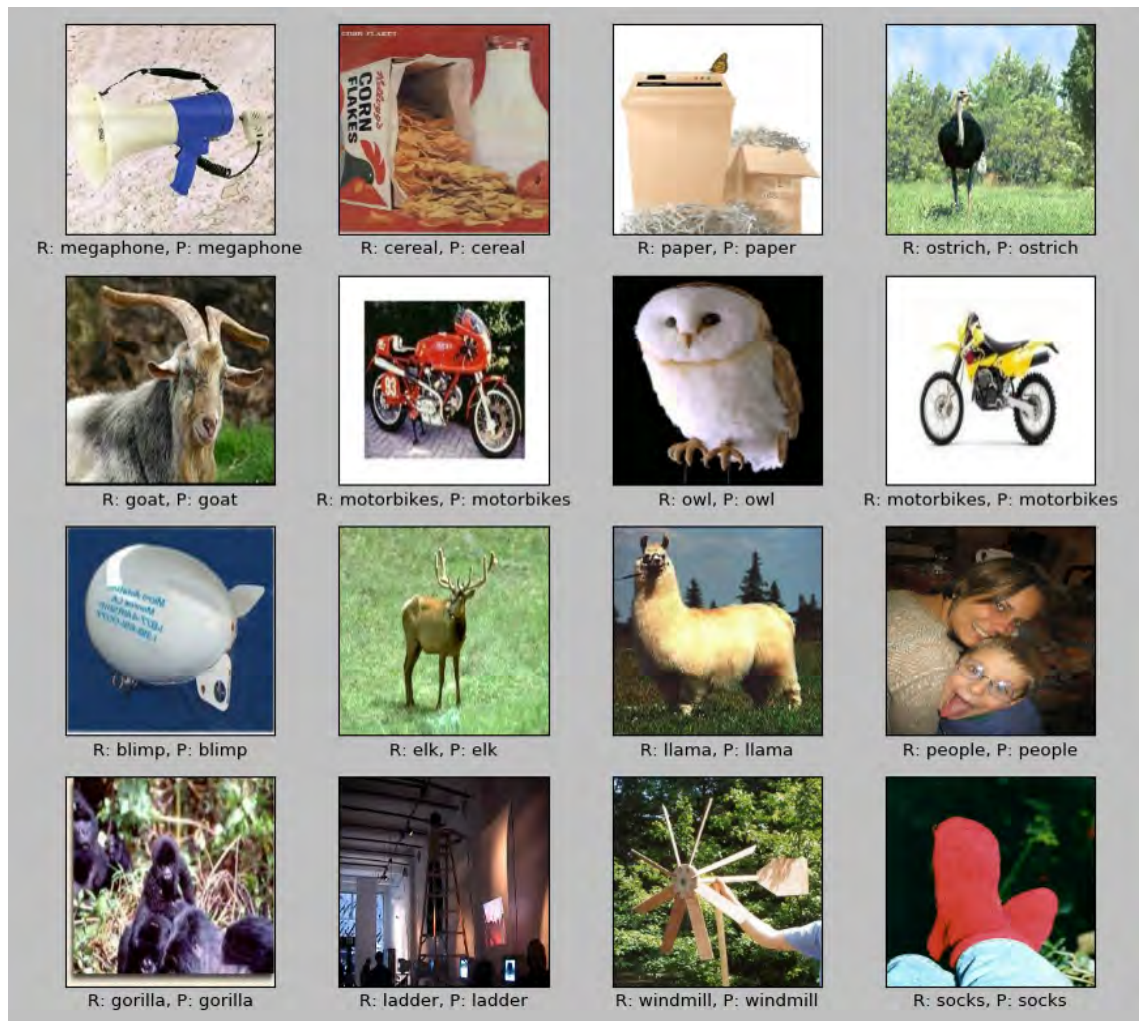


**Figura 4.10.** Rendimiento en entrenamiento en top 1 con la red ToniNet.



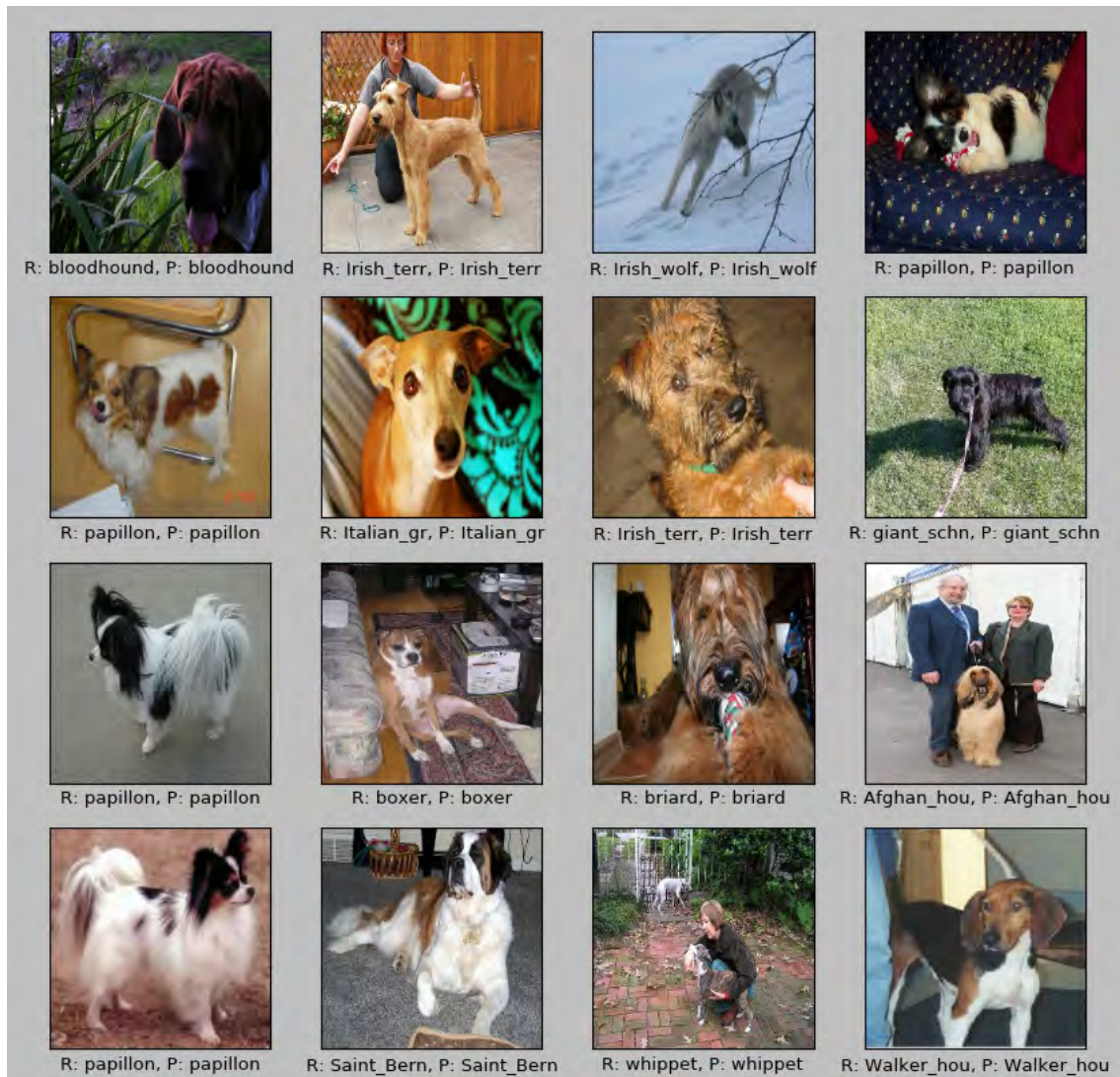
**Figura 4.11.** Rendimiento en entrenamiento en top 5 con la red ToniNet.





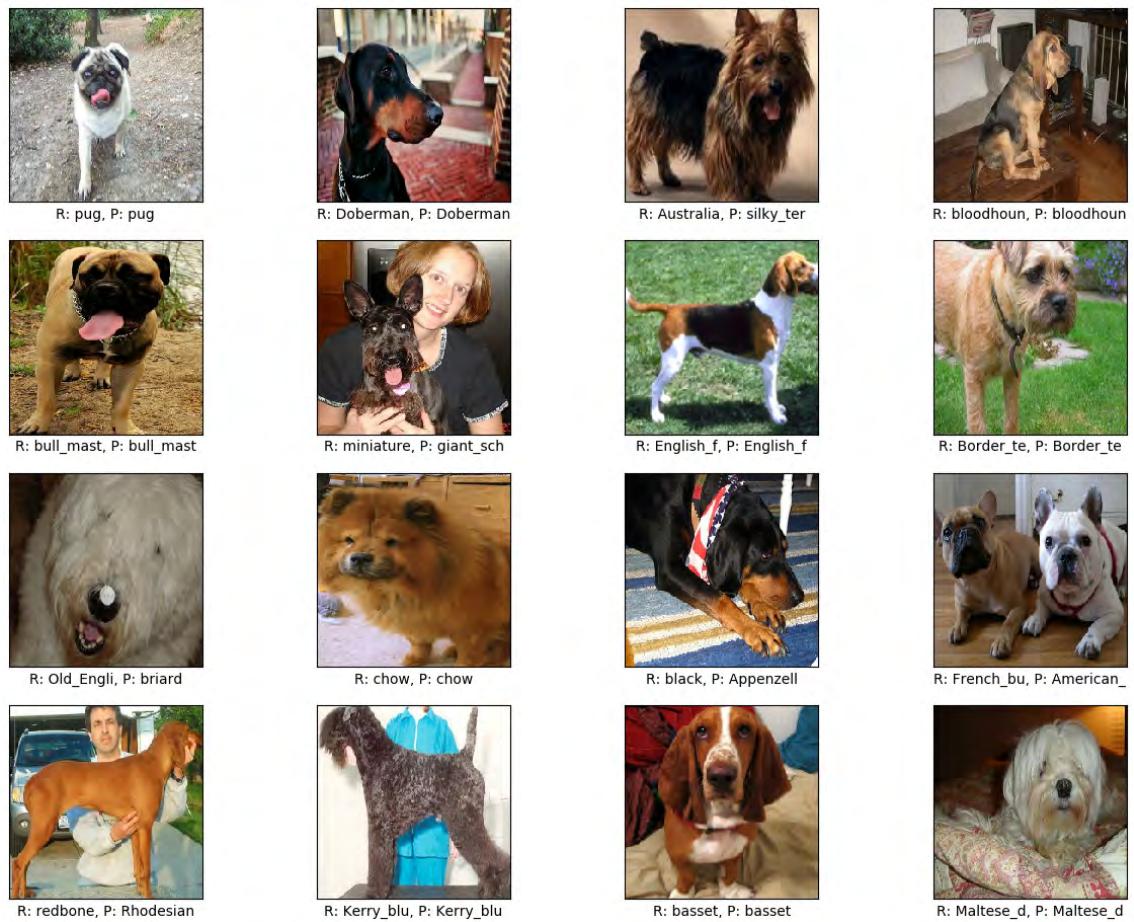
**Figura 4.12.** Entrenamiento con caltech, la letra *R* significa que es la clasificación real, la letra *P* significa que es la clasificación dada por la red.

#### 4. Resultados experimentales



**Figura 4.13.** Entrenamiento con Stanford Dogs, la letra *R* significa que es la clasificación real, la letra *P* significa que es la clasificación dada por la red.





**Figura 4.14.** Pruebas con Stanford Dogs, la letra *R* significa que es la clasificación real, la letra *P* significa que es la clasificación dada por la red.



# Capítulo 5

## Conclusiones y trabajo futuro

En este proyecto se ha mostrado que es posible construir una red neuronal convolucional para la clasificación de imágenes. Para lograr esto se ha descrito: la teoría, el análisis, el diseño, la construcción y la evaluación de la red. Después se hizo una comparación con otras redes famosas.

La red diseñada dio mejores resultados con el conjunto de imágenes de Oliva & Torralba y StanfordDogs comparándola con AlexNet, GoogleNet y ResNet 152. Fue superada por ResNet 152 con el conjunto de imágenes Caltech 256, pero el tiempo de entrenamiento de nuestra red fue menor y con mayor rendimiento que AlexNet y GoogleNet.

La ventaja de la red con respecto a las comparadas; es que tiene diferentes secciones de extracción de características, con esto necesita menos capas, como por ejemplo: ResNet 152 y GoogleNet. La desventaja principal es que necesita más tiempo de entrenamiento que AlexNet y GoogleNet para lograr mejores resultados pero con menor tiempo que ResNet.

Se ponen las bases para la construcción de redes usando diferentes secciones para la extracción de características. Se pueden añadir más secciones pero analizando la capacidad del hardware con que se cuenta. Las redes profundas como ResNet con 152 capas extraen las características solo de una sección, en cambio en nuestra red se pueden extraer características de múltiples secciones.

La red se puede mejorar en la etapa de clasificación, principalmente en las capas totalmente conectadas, también ampliando las secciones.

Para mejorar el rendimiento en pruebas se implementó la técnica de aumento de datos. Aunque disminuyó el rendimiento en el entrenamiento, en las pruebas

se obtuvieron mejores resultados. Las redes necesitan miles de imágenes para que puedan generalizar de buena forma; así, con el aumento de datos, simulamos tener más imágenes de las que contiene el conjunto de imágenes.

Se desarrolló un toolbox para separar los conjuntos de entrenamiento y prueba, logrando crear archivos de tipo tfrecord, estos archivos son usados para evitar el desbordamiento de memoria al realizar el entrenamiento, solo se extraen de manera aleatoria el lote necesario a analizar, ya que se trabajaron con miles de imágenes. Se pueden crear diferentes conjuntos de imágenes para entrenamiento y pruebas, logrando que al entrenar y evaluar siempre se tengan los mismos datos.

Utilizar múltiples GPUs nos ayuda a disminuir tiempo para el procesamiento de datos y son una herramienta posible de usarse con TensorFlow para la construcción de nuevas redes.

Este trabajo puede ayudar a las personas que estén en proceso de aprendizaje de las redes neuronales convolucionales para la clasificación de imágenes, describiendo el proceso de creación de una CNN para el entrenamiento y prueba.

Como trabajo futuro se planea utilizar la red para la detección de objetos y segmentación de imágenes. Después desarrollar aplicaciones para celulares y otros dispositivos en donde se pueda implementar la detección de objetos y segmentación. También se planea experimentar con más secciones; para trabajar con el color, la forma, la textura, entre otras.

# Bibliografía

- [1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.
- [2] G. E. H. Alex Krizhevsky, Ilya Sutskever, “Imagenet classification with deep convolutional neural networks,” *Neural Information Processing Systems (NIPS)*, 2012.
- [3] G. von Zitzewitz, “Survey of neural networks in autonomous driving,” 07 2017.
- [4] T. M. database of handwritten digits, “.” <http://yann.lecun.com/exdb/mnist/>, Julio 2017.
- [5] J. A. ANderson, *Redes neuronales*. Alfaomega, 1 ed., 2007.
- [6] C. Zhang, P. Patras, and H. Haddadi, “Deep learning in mobile and wireless networking: A survey,” *CoRR*, vol. abs/1803.04311, 2018.
- [7] NVIDIA, “NVIDIA TITAN X.” <https://www.nvidia.com/es-la/geforce/products/10series/titan-x>, Enero 2019.
- [8] S. Oh, M. Kim, D. Kim, M. Jeong, and M. Lee, “Investigation on performance and energy efficiency of cnn-based object detection on embedded device,” in *2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT)*, pp. 1–4, Aug 2017.
- [9] Y. J. P. S. S. R. Christian Szegedy, Wei Liu, “Going deeper with convolutions,” *IEEE Xplore*, 2015.
- [10] S. R. J. S. Kaiming He, Xiangyu Zhang, “Deep residual learning for image recognition,” *IEEE Xplore*, 2015.

- [11] B. T. Nugraha, S. Su, and Fahmizal, “Towards self-driving car using convolutional neural network and road lane detector,” in *2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*, pp. 65–69, Oct 2017.
- [12] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” *CoRR*, vol. abs/1312.6229, 2013.
- [13] A. Mollahosseini, D. Chan, and M. H. Mahoor, “Going deeper in facial expression recognition using deep neural networks,” in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1–10, March 2016.
- [14] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *CoRR*, vol. abs/1311.2524, 2013.
- [15] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 664–676, April 2017.
- [16] A. Prasoon, K. Petersen, C. Igel, F. Lauze, E. Dam, and M. Nielsen, “Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network,” 01 2013.
- [17] A. Khumaidi, E. M. Yuniarno, and M. H. Purnomo, “Welding defect classification based on convolution neural network (cnn) and gaussian kernel,” in *2017 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pp. 261–265, Aug 2017.
- [18] Tensorflow, “Tensorflow.” <https://www.tensorflow.org>, Enero 2017.
- [19] Oliva and Torralba, “.” <http://cvcl.mit.edu/database.htm>, Mayo 2016.
- [20] A. Khosla, Nityananda, Jayadevaprakash, B. Yao, and L. Fei-Fei, “Stanford Dogs Dataset.” <http://vision.stanford.edu/aditya86/ImageNetDogs/>, Septiembre 2017.

- 
- [21] Caltech256, “Caltech 256 Dataset.” [www.vision.caltech.edu/ImageDatasets/Caltech256](http://www.vision.caltech.edu/ImageDatasets/Caltech256), Mayo 2016.
- [22] S. V. Lab, “Large Scale Visual Recognition Challenge (ILSVRC).” <http://www.image-net.org/challenges/LSVRC/>, Enero 2017.
- [23] P. U. Stanford Vision Lab, Stanford University, “ImageNet.” <http://image-net.org/>, Octubre 2017.
- [24] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, “Convolutional neural networks for large-scale remote-sensing image classification,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, pp. 645–657, Feb 2017.
- [25] E. Nasr-Esfahani, S. Samavi, N. Karimi, S. M. R. Soroushmehr, M. H. Jafari, K. Ward, and K. Najarian, “Melanoma detection by analysis of clinical images using convolutional neural network,” in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 1373–1376, Aug 2016.
- [26] N. Kussul, M. Lavreniuk, A. Shelestov, and B. Yailymov, “Along the season crop classification in ukraine based on time series of optical and sar images using ensemble of neural network classifiers,” in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 7145–7148, July 2016.
- [27] P. Wang, L. Li, Y. Jin, and G. Wang, “Detection of unwanted traffic congestion based on existing surveillance system using in freeway via a cnn-architecture trafficnet,” in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1134–1139, May 2018.
- [28] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, “End to end learning for self-driving cars,” *CoRR*, vol. abs/1604.07316, 2016.
- [29] P. Wang, Q. Zhang, L. Li, F. Ru, D. Li, and Y. Jin, “Deep learning-based gesture recognition for control of mobile body-weight support platform,” in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1803–1808, May 2018.

- [30] K. Muhammad, J. Ahmad, I. Mehmood, S. Rho, and S. W. Baik, “Convolutional neural networks based fire detection in surveillance videos,” *IEEE Access*, vol. 6, pp. 18174–18183, 2018.
- [31] X. Liu, J. Li, C. Hu, and J. Pan, “Deep convolutional neural networks-based age and gender classification with facial images,” in *2017 First International Conference on Electronics Instrumentation Information Systems (EIIS)*, pp. 1–4, June 2017.
- [32] T. Suzuki, Y. Aoki, and H. Kataoka, “Pedestrian near-miss analysis on vehicle-mounted driving recorders,” in *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pp. 416–419, May 2017.
- [33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017.
- [34] S. Suriyal, C. Druzgalski, and K. Gautam, “Mobile assisted diabetic retinopathy detection using deep neural network,” in *2018 Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE/PAHCE)*, pp. 1–4, March 2018.
- [35] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes challenge: A retrospective,” *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan 2015.
- [36] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. G. Rodríguez, “A review on deep learning techniques applied to semantic segmentation,” *CoRR*, vol. abs/1704.06857, 2017.
- [37] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, pp. 1798–1828, Aug 2013.
- [38] K. Lin, H. F. Yang, J. H. Hsiao, and C. S. Chen, “Deep learning of binary hash codes for fast image retrieval,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 27–35, June 2015.
- [39] H. Lai, Y. Pan, Y. Liu, and S. Yan, “Simultaneous feature learning and hash



- coding with deep neural networks,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3270–3278, June 2015.
- [40] L. A. Gatys, A. S. Ecker, and M. Bethge, “A neural algorithm of artistic style,” *CoRR*, vol. abs/1508.06576, 2015.
- [41] J. Johnson, A. Alahi, and F. Li, “Perceptual losses for real-time style transfer and super-resolution,” *CoRR*, vol. abs/1603.08155, 2016.
- [42] J. Liao, Y. Yao, L. Yuan, G. Hua, and S. B. Kang, “Visual attribute transfer through deep image analogy,” *CoRR*, vol. abs/1705.01088, 2017.
- [43] D. Michie, D. J. Spiegelhalter, C. C. Taylor, and J. Campbell, eds., *Machine Learning, Neural and Statistical Classification*. Upper Saddle River, NJ, USA: Ellis Horwood, 1994.
- [44] S. A. Pooja Kamavisdar, Sonam Saluja, “A survey on image classification approaches and techniques,” *International Journal of Advanced Research in Computer and Communication Engineering*, January 2013.
- [45] D. Tian, “A review on image feature extraction and representation techniques,” *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, pp. 385–395, 01 2013.
- [46] G. Kumar and P. Bhatia, “A detailed review of feature extraction in image processing systems,” 02 2014.
- [47] “Embedding Projector.” <https://projector.tensorflow.org/>, Julio 2018.
- [48] D. Lu and Q. Weng, “A survey of image classification methods and techniques for improving classification performance,” vol. 28, pp. 823 – 870, 03 2007.
- [49] R. A. Española, “Diccionario de la lengua española,” vol. 21, 1992.
- [50] P. N. Stuart Russell, *Inteligencia Artificial un enfoque moderno*. Prentice Hall, 1 ed., 1996.
- [51] Abdullah and M. S. Hasan, “An application of pre-trained cnn for image classification,” in *2017 20th International Conference of Computer and Information Technology (ICCIT)*, pp. 1–6, Dec 2017.
- [52] M. Pérez-Ortiz, S. Jiménez-Fernández, P. A. Gutierrez, E. Alexandre, C. Hervás-Martínez, and S. Salcedo-Sanz, “A review of classification problems

- and algorithms in renewable energy applications,” 2016.
- [53] J. V. Cortez, ed., *Síntesis automática de memorias asociativas mediante programación genética*. 2009.
  - [54] L. Yang and M. Deng, “Based on k-means and fuzzy k-means algorithm classification of precipitation,” in *2010 International Symposium on Computational Intelligence and Design*, vol. 1, pp. 218–221, Oct 2010.
  - [55] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” vol. 521, pp. 436–44, 05 2015.
  - [56] B. H. Juang, “Deep neural networks – a developmental perspective,” *APSIPA Transactions on Signal and Information Processing*, vol. 5, p. e7, 2016.
  - [57] J. Schmidhuber, “Deep learning in neural networks: An overview,” *CoRR*, vol. abs/1404.7828, 2014.
  - [58] W. S. MCCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics Volume 5, 1943*, 1943.
  - [59] F. Roseblatt, “The perceptron, a perceiving and recognizing automaton,” *Corneell aeronautical laboratory, inc*, 1957.
  - [60] D. M. S. James A. Freeman, *Redes neuronales. Algoritmos, aplicaciones y técnicas de programación*. Addison-Wesley Iberoamericana, S. A., 1 ed., 1993.
  - [61] Y. Bengio, “Learning deep architectures for ai,” vol. 2, pp. 1–55, 01 2009.
  - [62] C. R. D. Shaima I. Jabbar, “Using convolutional neural network for edge detection in musculoskeletal ultrasound images,” *IEEE*, 2016.
  - [63] X. Jia, “A deep convolutional neural network for bleeding detection in wireless capsule endoscopy images,” *IEEE*, 2016.
  - [64] E. L. Riveros, J. G. Chávez, and J. C. G. Cáceres, “Analyzing the effect of hyperparameters in a automobile classifier based on convolutional neural networks,” in *2016 35th International Conference of the Chilean Computer Science Society (SCCC)*, pp. 1–7, Oct 2016.
  - [65] P. V. Yoshua Bengio, Aaron Courville, “Representation learning: A review and new perspectives,” *Arxiv:1206.5538v3*, 2014.

- 
- [66] D. Mishkin, N. Sergievskiy, and J. Matas, “Systematic evaluation of CNN advances on the imagenet,” *CoRR*, vol. abs/1606.02228, 2016.
- [67] R. E. W. Raffal c Gonzalez, “Digital image processing,” *Prentice Hall*, 2002.
- [68] D. Strigl, K. Kofler, and S. Podlipnig, “Performance and scalability of gpu-based convolutional neural networks,” in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp. 317–324, Feb 2010.
- [69] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
- [70] X. Qi, T. Wang, and J. Liu, “Comparison of support vector machine and softmax classifiers in computer vision,” in *2017 Second International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, pp. 151–155, Dec 2017.
- [71] J. van Doorn, “Analysis of deep convolutional neural network architectures,” 2014.
- [72] S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.04747, 2016.
- [73] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” *Proceedings of the 30th International Conference on Machine Learning*, 2013.
- [74] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014.
- [75] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1139–1147, PMLR, 17–19 Jun 2013.
- [76] K. Shi, H. Bao, and N. Ma, “Forward vehicle detection based on incremental learning and fast r-cnn,” in *2017 13th International Conference on Computational Intelligence and Security (CIS)*, pp. 73–76, Dec 2017.

- [77] H. Guo, S. Li, B. Li, Y. Ma, and X. Ren, “A new learning automata-based pruning method to train deep neural networks,” *IEEE Internet of Things Journal*, vol. 5, pp. 3263–3269, Oct 2018.
- [78] Z. Feng, Z. Sun, and L. Jin, “Learning deep neural network using max-margin minimum classification error,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2677–2681, March 2016.
- [79] S.-i. Amari, N. Murata, K.-R. Muller, M. Finke, and H. H. Yang, “Asymptotic statistical theory of overtraining and cross-validation,” *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 985–996, 1997.
- [80] A. M. Giancarlo Zaccone, Md. Rezaul Karim, ed., *Deep learning with TensorFlow*. Birmingham, UK: Packt Publishing, 2017.
- [81] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *CoRR*, vol. abs/1502.03167, 2015.
- [82] J. Ding, X. Kang, and X. Hu, “Validating a deep learning framework by metamorphic testing,” in *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*, pp. 28–34, May 2017.
- [83] NVIDIA, “Nvidia.” <http://la.nvidia.com/page/home.html>, Enero 2017.
- [84] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, “Fast convolutional nets with fbfft: A GPU performance evaluation,” *CoRR*, vol. abs/1412.7580, 2014.
- [85] D. S. Banerjee, K. Hamidouche, and D. K. Panda, “Re-designing cntk deep learning framework on modern gpu enabled clusters,” in *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 144–151, Dec 2016.
- [86] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang, “Tensorflow: A system for large-scale machine learning,” *CoRR*, vol. abs/1605.08695, 2016.
- [87] P. Goldsborough, “A tour of tensorflow,” *CoRR*, vol. abs/1610.01178, 2016.

- [88] NVIDIA, “Cuda toolkit download.” <https://developer.nvidia.com/>, Julio 2017.
- [89] NVIDIA, “Download drivers.” <http://www.nvidia.com/Download/index.aspx>, Julio 2017.
- [90] Ubuntu, “Download ubuntu desktop.” <https://www.ubuntu.com/download/desktop>, Julio 2017.



# Apéndice A

## Configuración y puesta en marcha de hardware y software

Antes de la construcción de la red e implementación se ha tenido que realizar lo siguiente:

### A.1. Instalación de software

Para este trabajo se decidió utilizar software libre, para la creación de la red neuronal.

Se utilizó TensorFlow por las siguientes características:

- Se tiene planeado realizar los entrenamientos y las pruebas con más de una GPU, TensorFlow los detecta automáticamente e incluye funciones para utilizar dispositivos como GPUs y CPUs dependiendo de lo que se necesite.
- TensorFlow provee TensorBoard para visualizar la estructura del grafo y los tensores, esto es bueno para ver el comportamiento del diseño de la arquitectura.
- Se busca como proyecto a futuro implementar el aprendizaje a dispositivos móviles, TensorFlow puede ser utilizado para dispositivos con Android.
- Existe una librería llamada TFLearn, construida sobre TensorFlow, hace más rápido el desarrollo de codificación ya que cuenta con funciones predefinidas que solo reciben parámetros de la red y da salidas de datos. Es recomendable utilizarlo para el desarrollo de una red de forma rápida, pero para este proyecto se pretende comprender lo que internamente hacen las funciones, se podrá estar utilizando a la par para realizar comparaciones.

- Se puede utilizar numpy, cv2, math, entre otras librerías destacadas para operaciones entre arreglos multidimensionales, procesamiento de imágenes, entre otras ventajas.

Una vez seleccionado el ambiente de desarrollo, se puede iniciar la instalación, pero antes de hacerlo es necesario definir si se cuenta con una GPU por lo que debe de cumplir con ciertas características para poder utilizarlo, estas son las siguientes:

- Tensorflow requiere utilizar GPUs NVIDIA (se inició el proyecto con una Laptop Alienware con 16 GB de memoria RAM, procesador i7-6700HQ, incluye una GPU 980 con 4 GB de memoria RAM).
- Instalar CUDA Toolkit 8.0, se puede descargar de la página oficial [88].
- Para utilizar la GPU, se necesitan los drivers actualizados, estos pueden ser bajados de la página de NVIDIA [89] y que sean compatibles con CUDA Toolkit 8.0.
- cuDNN, provee rutinas para realizar convolución, pooling, normalización y capas de activación. Utilizadas principalmente para aprendizaje profundo.
- Librería Libcupti-dev, librería necesaria para trabajar con el sistema operativo Ubuntu, para este trabajo se utiliza Ubuntu 16.04 [90].

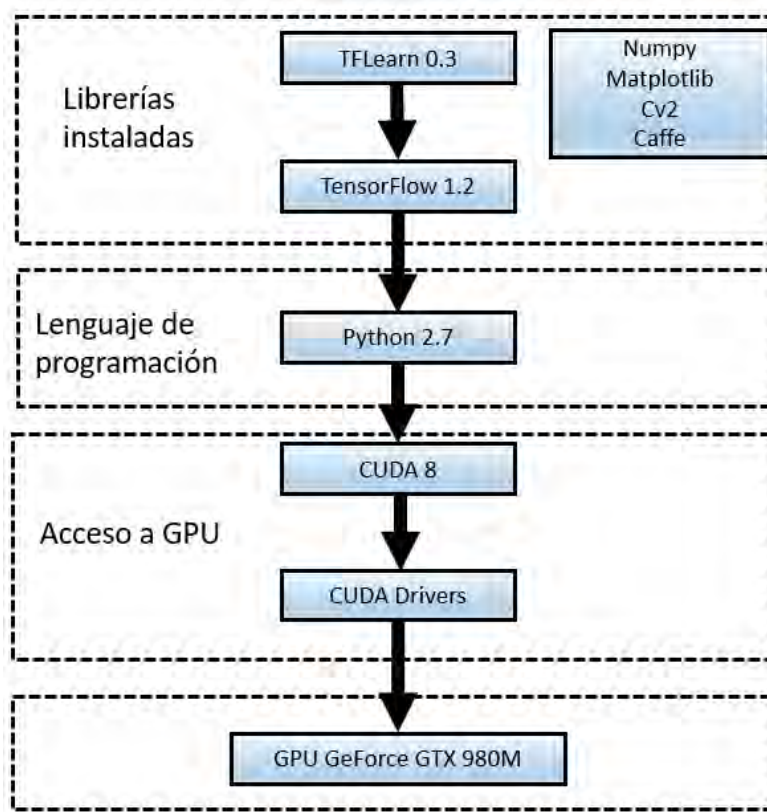
Se puede utilizar TensorFlow con C++ o con Python, se opta por la segunda opción por la variedad de librerías que se pueden importar y utilizar. Hay dos diferentes formas de instalación, una es para Python 2.7 y la otra es para 3.4 a 3.6, descritas en la página oficial de TensorFlow, se tiene que tener presente para evitar errores al instalar. De forma general el uso de lo instalado para acceder a la GPU se muestra en la figura A.1.

## A.2. Requerimientos técnicos

### Hardware

- NVIDIA GeForce GTX TITAN X con 12 GB de memoria RAM, Procesador Intel(R) Xeon(R), 16 Gb de Memoria RAM, 1 disco duro de 500 GB.
- CPU: Intel Core i7-6800K @ 3.40GHz × 12





**Figura A.1.** Uso del software instalado para acceder a la GPU.

- GPU: NVIDIA GeForce GTX 1080/PCIe/SSE2

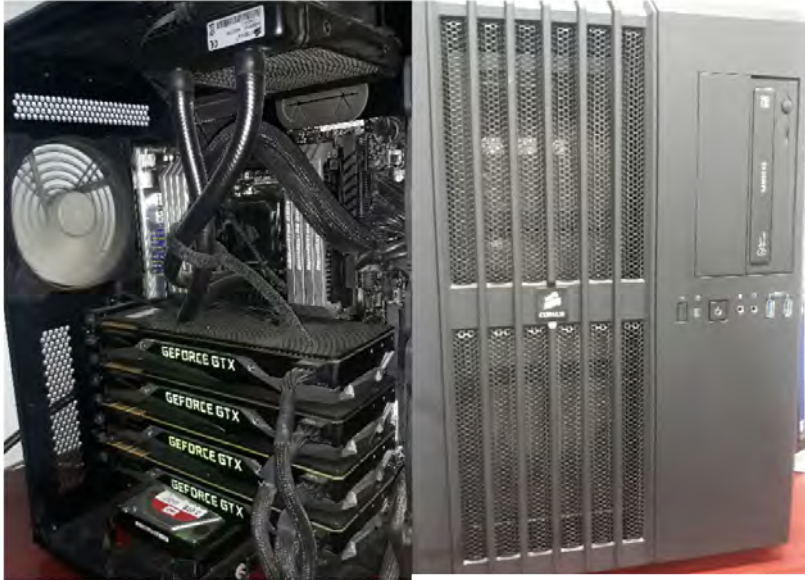
## Software

- OS: Linux Ubuntu 16.04.1 LTS 64-bit, linux kernel 4.12
- Python 2.7.12
- Tensorflow 1.4.1
- Opencv: 3.2.0
- Tkinter.
- Pip
- Matplotlib
- NumPy
- NVIDIA CUDA® 8.0
- NVIDIA cuDNN v5.1
- NVIDIA toolkits

## A. Configuración y puesta en marcha de hardware y software

---

En la figura A.2 se muestra un servidor con 4 GPUs utilizados para el entrenamiento de las redes.



**Figura A.2.** Servidor con 4 GPUs GTX 1080, utilizados para el entrenamiento de las redes.

# Apéndice B

## Toolbox para generar conjuntos de entrenamiento y prueba

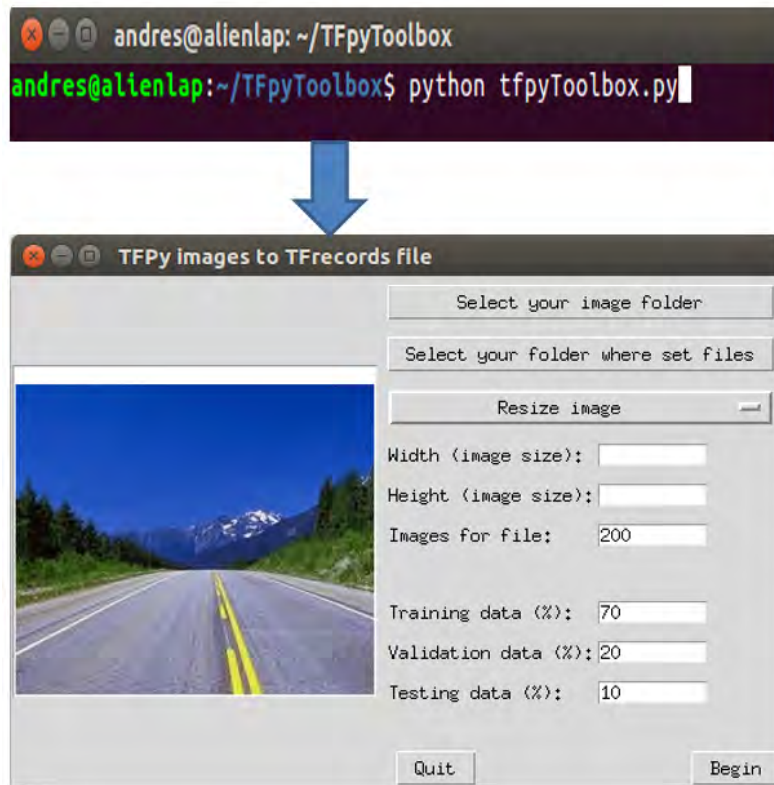
Para generar los conjuntos de entrenamiento y prueba en bases de datos de gran escala, como ImageNet, es recomendable generar archivos de tipo *tfrecord*. Archivos que son utilizados para guardar una gran cantidad de imágenes, que pueden ser de diferentes tamaños, codificadas en arreglos multidimensionales.

Para generar los archivos *tfrecord*, se desarrollo un toolbox, que permite crear los archivos de entrenamiento y prueba de una manera fácil y rápida; además de permitir la generación de parámetros que ayudan al entrenamiento de la red. El toolbox se ejecuta directamente desde la línea de comandos con `python tfpyToolbox.py`, ver figura B.1. Cuando se visualiza la ventana podemos seleccionar la ubicación de los imágenes ya clasificadas, la ruta donde se guardarán los archivos, si se requiere redimensionar la imagen o recortar, el tamaño de la imagen, el tamaño de los conjuntos de imágenes como se muestra en la figura B.2. Nos genera diferentes carpetas con archivos de tipo *tfrecord* y los archivos de texto que nos ayuda en el entrenamiento como son todas las clases, tamaño de la imagen, totales por dataset, como se muestra en la imagen B.3.

La ventaja de este toolbox es que podemos generar conjuntos de entrenamiento y prueba, preparados para ser utilizados para cualquier CNN. El toolbox genera diferentes carpetas con archivos de tipo *tfrecord* y archivos de texto que ayuda en el entrenamiento como son: la descripción de todas las clases, tamaño de la imagen, totales por conjunto de datos. El toolbox separa la base de datos de entrenamiento para utilizarla con diferentes modelos y diferentes parámetros sin necesidad de volver

## B. Toolbox para generar conjuntos de entrenamiento y prueba

---



**Figura B.1.** Inicio TFPyToolbox.

a generar los conjuntos de datos, ésta separación se hace por cada clasificación, debido a que las clasificaciones tienen diferentes cantidades de imágenes.

De los archivos *tfrecord* se leen las imágenes y las clases separándolos por lotes y de manera aleatoria, *Tensorflow* tiene la función *shuffle\_batch* para hacerlo, esta función regresa dos lotes, uno de imágenes y otro de clases en nuestro caso será un lote de 32. Cuando se obtienen las imágenes se pueden visualizar algunas para verificar que están de forma correcta y poder continuar con el proceso, de manera general se muestra el proceso de generación y lectura de los archivos en la figura B.4.

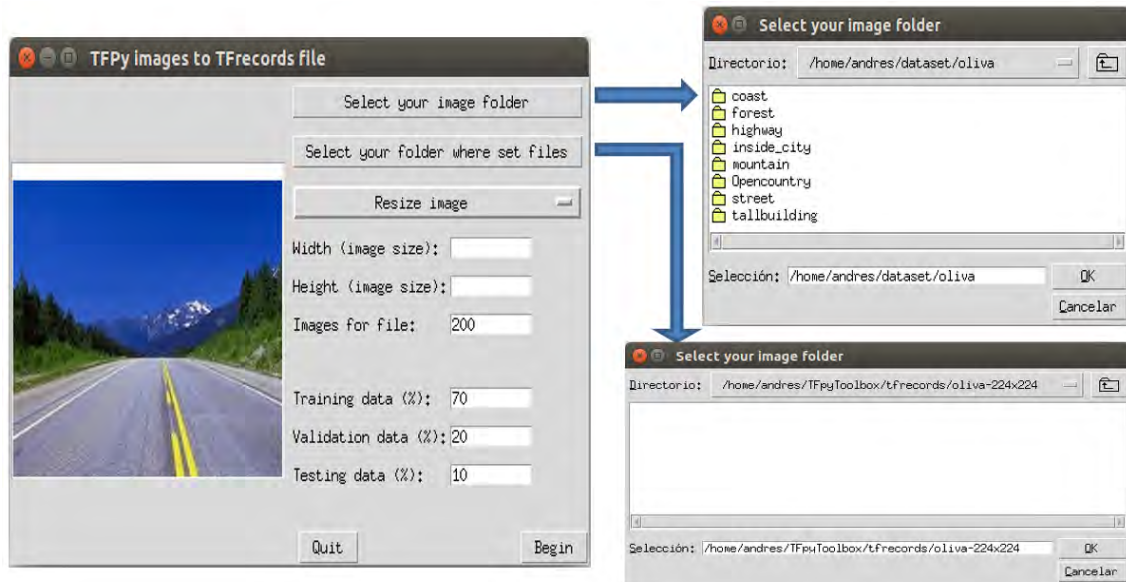


Figura B.2. Selección del conjunto de imágenes con TFPyToolbox.

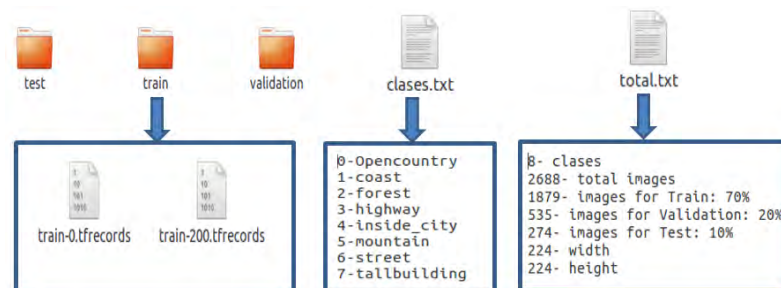


Figura B.3. Generación de dataset.

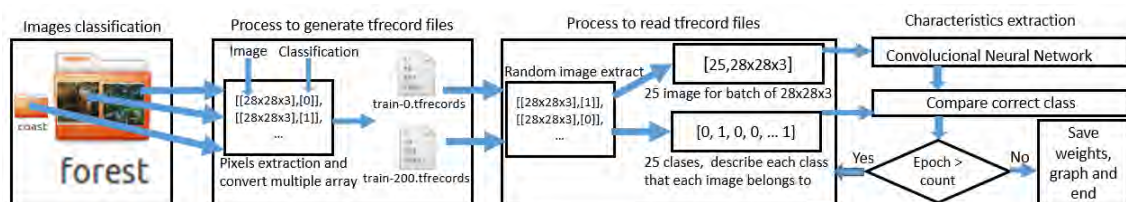


Figura B.4. Escritura y lectura de los archivos tfrecord.



# Apéndice C

## Artículos

- Fidel Lopez-Saca, Carlos Avilés-Cruz, Andrés Ferreyra-Ramírez, Juan Villegas-Cortez. Red neuronal convolucional con extracción de características multi-columna para clasificación de imágenes. Research in Computing Science (Latindex). Artículo aceptado para publicación en el XI Congreso Mexicano de Inteligencia Artificial “COMIA 2019”.
- Fidel López Saca, Andrés Ferreyra Ramírez, Carlos Avilés Cruz. Prototipo funcional para clasificación de imágenes con salida de audio en un sistema embebido con red neuronal convolucional . Pistas Educativas (Latindex), pp. 804-821. Vol. 40, Num. 130, Noviembre 2018. ISSN 2448-847X.
- Fidel Lopez-Saca, Carlos Avilés-Cruz, Andrés Ferreyra-Ramírez, Juan Villegas-Cortez, Arturo Zuñiga-López and Eduardo Rodriguez-Martinez. Preprocesamiento de bases de datos de imágenes para mejorar el rendimiento de redes neuronales convolucionales. Research in Computing Science (Latindex), Vol. 147(7), pp. 35-45. 2018. ISSN 18704069.

# Red neuronal convolucional con extracción de características multi-columna para clasificación de imágenes

Fidel López Saca, Andrés Ferreyra Ramírez, Carlos Avilés Cruz, and Juan Villegas Cortéz

Universidad Autónoma Metropolitana, Unidad Azcapotzalco, Ciudad de México, México

`fidelosmcc@gmail.com`, `{fra, caviles, juanvc}@correo.azc.uam.mx`

**Resumen.** En los últimos años, las redes neuronales convolucionales han traído una evolución significativa en la clasificación de imágenes, estos métodos poderosos de aprendizaje automático extraen características de nivel inferior y las envían a las siguientes capas para identificar características de nivel superior que mejoran el rendimiento. En el presente artículo se propone una arquitectura de una red convolucional con un enfoque que logra incrementar el rendimiento de clasificación, es una estructura jerárquica que es fácil de construir, es adaptable y fácil de entrenar con un buen rendimiento en tareas de clasificación de imágenes. En la arquitectura propuesta, la imagen ingresa a la red por tres secciones de extracción de características que utilizan diferentes filtros de convolución, las características extraídas son combinadas y enviadas a capas totalmente conectadas para realizar la clasificación. Los resultados experimentales muestran que la arquitectura propuesta tiene un rendimiento superior a diferentes redes convolucionales tradicionales tales como AlexNet, GoogleNet y ResNet 152.

**Palabras clave:** Redes Neuronales Convolucionales · Clasificación · Extracción de características.

## Convolutional neural network with extraction of multi-column features for image classification

**Abstract.** In recent years, convolutional neural networks have brought a significant evolution in the classification of images, these powerful methods of machine learning extract lower level features and send them to the following layers to identify higher level features that improve performance.

This article proposes an architecture of a convolutional network with an approach that manages to increase classification performance, is a hierarchical structure that is easy to build, is adaptable



and easy to train with a good performance in image classification tasks. In the proposed architecture, the image enters the network through three extraction sections of characteristics that use different convolution filters, the extracted characteristics are combined and sent to totally connect layers to perform the classification. The experimental results show that the proposed architecture has a superior performance for different traditional convolutional networks such as Alexnet, Googlenet and Resnet 152.

**Keywords.** Convolutional Neural Network, Classification, Feature extraction.

## 1 Introducción

En las últimas décadas, el crecimiento constante de imágenes digitales -como fuente principal de representación de la información para aplicaciones científicas- ha hecho de la clasificación de imágenes una tarea desafiante. Con el fin de alcanzar rendimientos de clasificación altos, se han propuesto diferentes técnicas de reconocimiento de patrones, entre las que se encuentran los métodos de aprendizaje profundo que hoy en día son un foco de estudio en el procesamiento de imágenes y visión por computadora. En este enfoque, la arquitectura más popular para la tarea de clasificación de imágenes son las redes neuronales convolucionales (CNNs, por sus siglas en inglés); una red construida de múltiples capas y en donde cada capa modela un campo receptivo de la corteza visual lo que la hace mucho más efectiva en tareas de visión artificial [11].

Las CNN combinan las características de bajo nivel dentro de características abstractas de alto nivel con transformaciones no lineales, lo que le permite tener la capacidad de aprender la representación semántica de las imágenes. Estas redes extraen características generalmente útiles de los datos con o sin etiquetas, detectan y eliminan redundancias de entrada y preservan solo aspectos esenciales de los datos en representaciones sólidas y discriminativas [2]; pueden capturar las características mas obvias de los datos [19], por lo que podrían lograr mejores resultados en varias aplicaciones. A diferencia de las características creadas a mano, como SIFT [12] y HOG [3]; las características extraídas por la CNN se generan de extremo a extremo, lo que elimina la intervención humana. Las CNN tienen menos conexiones y parámetros lo que favorece que la extracción de características sea más eficiente.

La mayoría de arquitecturas de redes convolucionales tales como AlexNet [10], GoogleNet [16], VGG [15], ResNet 152 [8], y muchas otras [7], [18], [13], [6], [5], utilizan el mismo concepto para producir mapas de características en las capas de convolución, seguidas de capas de ‘pooling’ para reducir la dimensión de los mapas y conforme profundizan en la arquitectura, duplican el número de filtros para compensar la reducción a la mitad del tamaño de los mapas de características posteriores. La profundidad de la red favorece el rendimiento de clasificación y evita la desaparición de los gradientes mediante el uso de la in-

ferencia de clases en capas de convolución consecutivas y la capa de pooling máximo, o el uso de capas softmax que realiza el desvanecimiento de los gradientes [16], [15],[19]. Algunas de estas arquitecturas, utilizan nuevas funciones de activación, métodos de regularización de actualización de pesos, inferencias de clase, entrenamiento previo por capas en enfoques supervisados; mostrado resultados muy prometedores [18], [5].

Aumentar el número de capas de una CNN significa aumentar la profundidad y el número de parámetros de la red; lo que complica el entrenamiento y reduce considerablemente el rendimiento, sobretodo con bases de datos pequeñas. Por otra parte, debido a la falta de características únicas, la fusión de características es cada vez mas importante para tareas como la clasificación y la recuperación de imágenes. Estas son técnicas que simplemente concatenan un par de características diferentes o utilizan métodos basados en el análisis de correlación canónica para la reducción de la dimensionalidad conjunta en el espacio de características.

En este artículo, se propone una red neuronal convolucional con extracción de características multi-columna para clasificación de imágenes. La red integra la capacidad de abstracción de las redes neuronales profundas y la capacidad de concatenar diferentes características. La red crece tanto en profundidad como en amplitud, la imagen a clasificar ingresa a través de tres secciones diferentes de extracción de características con diferentes filtros en las operaciones de convolución, las características extraídas son entonces concatenadas e ingresadas a capas totalmente conectadas que realizan la etapa de clasificación.

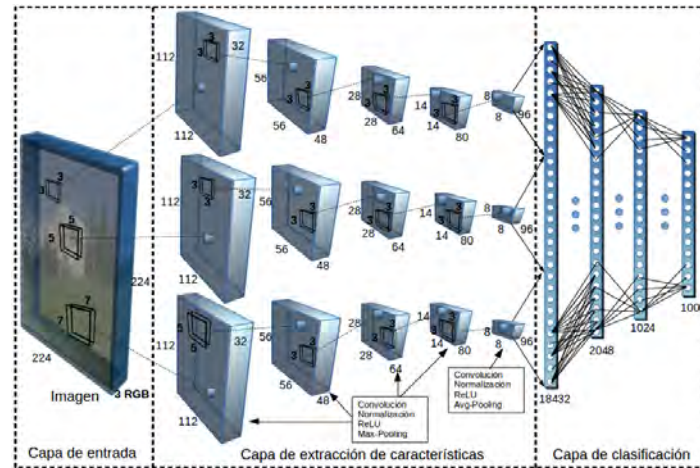
Los resultados muestran que la red tiene alto rendimiento en conjunto de imágenes como Oliva & Torralba [14], Stanford Dogs [9] y Caltech 256 [1]. Se realiza una comparación de la red propuesta con las redes existentes: AlexNet [10], GoogleNet [16] y ResNet [8], cuyas características están descritas en [4].

El documento está estructurado de la siguiente manera: En la sección 2 se describe el diseño de la red. La sección 3 analiza los conjuntos de datos empleados para los experimentos. En la sección 4 se describen los parámetros de entrenamiento. La sección 5 afirma la validez de la arquitectura propuesta mediante los experimentos realizados y los resultados experimentales. Finalmente en la sección 6, se discuten las conclusiones y el trabajo futuro.

## 2 Arquitectura propuesta

Aunque una sola CNN puede trabajar bien en problemas de clasificación de imágenes, puede presentar problemas para alcanzar precisiones altas en la fase de predicción. En la etapa de entrenamiento la red puede presentar un problema estadístico ya que el algoritmo de aprendizaje está buscando un espacio de parámetros (pesos y bias) que puede ser demasiado grande con respecto a la cantidad de datos de entrenamiento. En estos casos, puede haber muchos conjuntos de parámetros diferentes que produzcan la misma precisión, por lo que el algoritmo elige una de estas opciones; sin embargo, se corre el riesgo de que los parámetros elegidos no tengan buenas predicciones con las datos de prueba. En

muchas aplicaciones, el algoritmo de aprendizaje de la CNN, no puede garantizar encontrar el mejor conjunto de pesos y bias; por lo que se produce un problema computacional, ya que se tiene que jugar con el ajuste de las variables de entrenamiento de la red, tales como: número de épocas, tasa inicial de aprendizaje, tamaño del lote, etc. Por otra parte, cuando el espacio de soluciones no contiene un conjunto de parámetros que sea una aproximación correcta a la función objetivo verdadera, la red puede tener problemas en la representación de las características de las imágenes lo que dificulta aun más la clasificación y predicción. Los problemas mencionados anteriormente se pueden superar resolviendo el mismo problema de clasificación con varias CNN con arquitectura diferente y combinando los resultados. Esta expectativa nos llevó a proponer una arquitectura que en lugar de estar compuesta por diferentes CNN, esta diseñada con diferentes secciones de extracción de características; las cuales se combinan para realizar la clasificación. La arquitectura propuesta se muestra en la figura 1, está compuesta de 18 capas, las primeras quince son para la extracción de características y las últimas tres son para la clasificación. A continuación se hace una descripción de las capas de la red:



**Fig. 1.** Arquitectura general de la red neuronal propuesta.

- **Capa de entrada de la imagen:** esta capa establece una etapa de pre-procesamiento de las imágenes de entrada a la red. En esta capa las imágenes se pueden redimensionar, rotar e incluso se pueden tomar pequeñas muestras al azar. Esta capa esta diseñada para que la red acepte imágenes con dimensiones  $224 \times 224$  con una profundidad de tres, ya que la imagen ingresa con el formato RGB (ver figura 1, parte izquierda).
- **Capas de extracción de características:** la imagen ingresa por tres diferentes secciones de extracción de características, cada una extrae diferentes

características utilizando filtros de diferentes tamaños. La primera sección utiliza un filtro de  $3 \times 3$  en sus cinco capas. La segunda sección utiliza en la primera capa un filtro de  $5 \times 5$  y en las siguientes cuatro, filtros de  $3 \times 3$ . En la tercera sección, la primera capa utiliza un filtro de  $7 \times 7$  en la segunda uno de  $5 \times 5$  y en las últimas tres, filtros de  $3 \times 3$ . Las primeras cuatro capas de cada sección de extracción utilizan convolución, normalización, ReLU y max-pooling con un filtro de  $3 \times 3$  y paso de 2 con relleno. En la última capa de cada sección se utilizan convolución, normalización, ReLU y avg-pooling con un filtro de  $7 \times 7$ , un paso de uno sin relleno (ver figura 1, parte central).

- **Clasificación:** la salida de cada sección de extracción de características se concatena para generar un vector unidimensional. Así se puede continuar con las capas totalmente conectadas para realizar la clasificación, como se describe a continuación (ver figura 1, parte derecha).
  - **Concatenación de las salidas de las 3 secciones de extracción de características:** la clasificación inicia con la concatenación de la salida de cada sección de extracción, se tendrá un vector con dimensiones  $18432 \times 1$  (características de la imagen).
  - **Primera capa totalmente conectada:** tiene una profundidad de 2048 neuronas y es seguida de una capa de ReLU y una capa de Dropout.
  - **Segunda capa totalmente conectada:** consta de 1024 neuronas y es seguida de una capa de ReLU y una capa de Dropout.
  - **Tercera capa totalmente conectada:** esta capa es utilizada para ajustar la red convolucional a cada uno de los conjuntos de entrenamiento utilizados ya que es necesario igualar el número de neuronas de salida al número de clases de cada conjunto de entrenamiento. Esta capa es seguida de una capa de Softmax, una función de regresión, que ayuda a clasificar múltiples categorías.
- **Capa de salida:** es la capa final que se encarga de mostrar el porcentaje de éxito de clasificación.

Una red convolucional puede tener la tendencia a memorizar datos de entrenamiento, fenómeno conocido como sobreajuste, presentando porcentajes bajos de generalización. Para evitar esto, la red propuesta utiliza una capa de regularización comúnmente llamada “Dropout” en las primeras dos capas completamente conectadas. Las capas de Dropout hacen que la red sea más robusta a los datos de entrada imprevistos, y solo están activas durante la etapa de entrenamiento de la red, es decir, no están presentes durante la etapa de predicción.

### 3 Conjuntos de datos utilizados

En este trabajo se utilizaron 3 conjuntos de datos diferentes, los cuales se describen brevemente a continuación:

- **Oliva & Torralba [14]:** este conjunto de datos está compuesto por 2,688 imágenes de escenas a color que pertenecen a la misma categoría semántica.

La base de datos tiene un total de 8 categorías y las imágenes fueron obtenidas de diferentes fuentes: bases de datos comerciales, sitios web y cámaras digitales.

- **Stanford Dogs** [9]: este conjunto de datos consta de 20,580 imágenes a color, pertenecientes a 120 clases o razas de perros de todo el mundo. Este conjunto de datos se ha creado utilizando imágenes y anotaciones de ImageNet para la tarea de categorización de imágenes de grano fino.
- **Caltech 256** [1]: este conjunto de datos consta de 30,607 imágenes a color pertenecientes a 256 categorías más una de nombre “clutter” que contiene múltiples escenas. Cada categoría contiene de 80 a 827 imágenes y la mayoría de las categorías tiene alrededor de 100 imágenes.

En la tabla 1 se muestran algunas estadísticas de los conjuntos de datos.

**Tabla 1.** Conjuntos de datos utilizados y sus estadísticas.

Conjunto de datos	Clases	Dimensiones			No. imágenes		
		Ancho	Alto	Prof	Total	Min x clase	Max x clase
Oliva & Torralba	8	256	256	3	2,688	260	410
Stanford Dogs	120	200 – 500	150 – 500	3	20,580	148	252
Caltech 256	257	300	200	3	30,607	80	827

## 4 Parámetros de entrenamiento

Tanto la red propuesta como las redes con las que se hace la comparación: AlexNet, GoogleNet y ResNet, fueron entrenadas utilizando el algoritmo de optimización ADAM (adaptive moment estimation), con un tamaño de lote de  $\beta = 32$  imágenes y un decaimiento de pesos (factor de regularización) de  $\lambda = 0.0005$ . Los pesos iniciales en cada una de las capas fueron inicializados con una distribución gaussiana con una media de 0 y una desviación estándar de 0.01. Los umbrales de activación en cada una de las capas fueron inicializados a cero. Se inicio con una tasa de aprendizaje de  $\mu = 0.001$  la cual se disminuyó en un factor de 10 después de cada 50 épocas, para tener cambios de aprendizaje más específicos en 250 épocas de entrenamiento. Las redes fueron entrenadas en 4 GPUs NVIDIA GTX 1080, con 8 GB de memoria RAM y 2560 núcleos, con sistema operativo Linux Ubuntu 16.04, Linux kernel 4.12, Python 2.7, TensorFlow 1.12, NVIDIA CUDA®8.0, NVIDIA cuDNN v5.1.

La falta de una cantidad suficiente de imágenes de entrenamiento, el desequilibrio de imágenes por clase y el balance desigual de clases dentro de cada conjunto de datos, puede provocar que una red se sobreajuste. Para hacer a las redes más robustas e invariantes a transformaciones en los datos, mejorar la eficiencia del proceso de entrenamiento e incrementar la generalización; utilizamos



**Fig. 2.** Ejemplo de imágenes modificadas para el ingreso a la CNN.

el método de aumento de datos como una forma de crear imágenes nuevas con diferentes orientaciones [17].

Ya que las bases de datos contienen imágenes de diferentes tamaños, se hace un recorte aleatorio de la imagen de entrenamiento, el recorte es del mismo tamaño que acepta la capa de entrada de la red; es decir, de  $224 \times 224 \times 3$ . La imagen de entrada es reflejada de manera horizontal, de izquierda a derecha, con un 50% de probabilidad. El brillo y el contraste de la imagen se ajustan de manera aleatoria en intervalos del 63% y  $0.2 - 1.8$ , respectivamente. Finalmente la imagen es normalizada para que la red tenga cierta independencia de las propiedades de la imagen. En la figura 2, se muestran algunos ejemplos de imágenes modificadas con el aumento de datos.

## 5 Experimentos y resultados

Para evaluar el rendimiento, las redes se entrenaron desde cero con cada una de las bases de datos y se utilizó aumento de datos. Los conjuntos de entrenamiento fueron formados con el 70% de las imágenes de cada base de datos y el 30% restante se utilizó para formar los conjuntos de prueba; la selección de las imágenes se realizó de manera aleatoria. Para separar los conjunto de entrenamiento y prueba, se utilizó el toolbox desarrollado en [4], para crear archivos de tipo tfrecord para evitar el desbordamiento de la memoria al realizar el entrenamiento. En la tabla 2 se muestran las características de cada conjunto de datos utilizadas para realizar los experimentos en este proyecto.

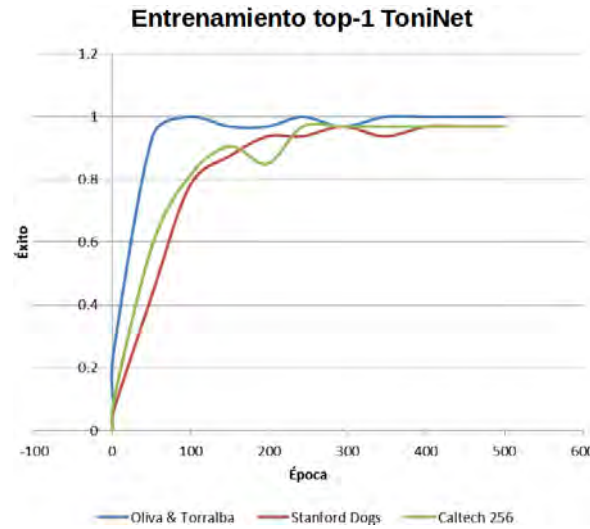
Para describir la precisión de las redes en la tarea de clasificación, utilizamos los términos **top-1** y **top-5**. El número **top-1** es la cantidad de veces que la etiqueta correcta tiene la probabilidad más alta predicha por la red. El número **top-5** es la cantidad de veces que la etiqueta correcta está dentro de las 5 clases principales predichas por la red.

**Tabla 2.** Conjuntos de imágenes separadas en entrenamiento y pruebas, utilizando el 70% de imágenes por clase para entrenamiento y el 30% para pruebas.

Conjunto de datos	Clases	Cantidad de imágenes		
		Entrenamiento	Prueba	Total
Oliva & Torralba	8	1,879	809	2,688
Stanford Dogs	120	14,358	6,222	20,580
Caltech 256	257	21,314	9,293	30,607

### 5.1 Evaluación de la red propuesta

En la figura 3 se muestran los resultados de entrenamiento de la red propuesta, la cual a partir de este momento llamaremos ToniNet para diferenciarla de las otras redes. Note que la red llega a una exactitud del 100% para el conjunto de entrenamiento Oliva & Torralba, mientras que para los conjuntos Stanford Dogs y Caltech 256 alcanza solo el 98%.

**Fig. 3.** Resultados de precisión top-1 en el entrenamiento de la red ToniNet.

Para evitar el sobre ajuste de la red ToniNet, se incluyeron capas de regularización (Dropout) en las dos primeras capas completamente conectas y se utilizó el aumento de datos. En la figura 4, se muestra los resultados obtenidos en la evaluación del sobreajuste de la red. Note que el error en la fase de entrenamiento disminuye de manera constante conforme aumenta el número de épocas, lo que refleja el crecimiento constante del rendimiento (Exactitud) de aprendizaje de la red.

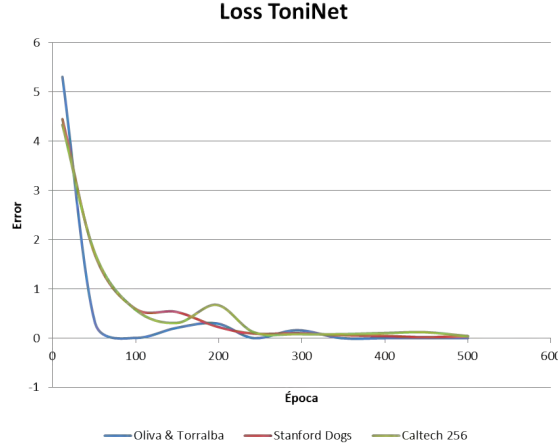


Fig. 4. Error de entrenamiento para la red ToniNet.

## 5.2 Comparación

En la tabla 3 se muestra un resumen de los resultados para la etapa de prueba de las redes AlexNet, GoogleNet, ResNet 152 y ToniNet. Note que ToniNet supera a AlexNet y a GoogleNet tanto en top-1 como en top-5 para los 3 conjuntos de prueba; sin embargo, ToniNet es superada por ResNet 152 solo con el conjunto de prueba Caltech 256. GoogleNet supera en precisión a AlexNet ya que es una red con mayor profundidad; es decir, es una red que cuenta con un número mayor de capas de extracción; por la misma razón, ResNet 152 supera a GoogleNet. Sin embargo, aunque ResNet 152 también tiene una profundidad mayor a ToniNet, está solo supera a ToniNet en precisión con Caltech 256; lo que enfatiza tanto la ventaja como la importancia de tener tres secciones de extracción de características en paralelo idénticas pero con filtros de diferentes tamaños.

Tabla 3. Resultados y comparación de redes

CNN	Oliva & Torralba			Stanford Dogs			Caltech 256		
	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5
AlexNet	19	90.4%	99.8%	175	43.2%	70.8%	213	57.5%	74.7%
GoogleNet	14	91.5%	99.8%	164	51.9%	80.2%	196	60.6%	79.0%
ResNet 152	85	92.8%	99.8%	538	53.6%	82.7%	739	64.7%	81.8%
ToniNet	34	94.6%	100%	255	55.2%	84.7%	371	62.5%	80.8%

En la tabla 3 también se muestran los tiempos de entrenamiento de las redes. GoogleNet es la red con el menor tiempo de entrenamiento seguida de AlexNet, ToniNet y ResNet 152. Tomando como referencia el mayor tiempo de entrenamiento, que corresponde a ResNet 152, para las tres bases de datos podemos comentar que en promedio: ToniNet tiene un tiempo de entrenamiento del



18.03% mayor a AlexNet y un 21.3% mayor a GoogleNet, pero un 54.13% menor a ResNet 152.

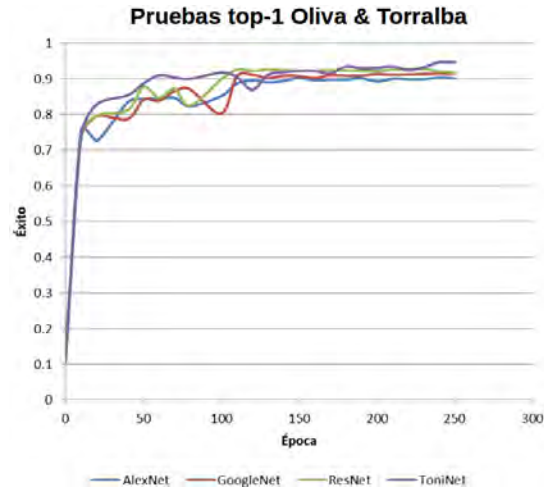


Fig. 5. Resultados de precisión top-1 (Oliva & Torralba).

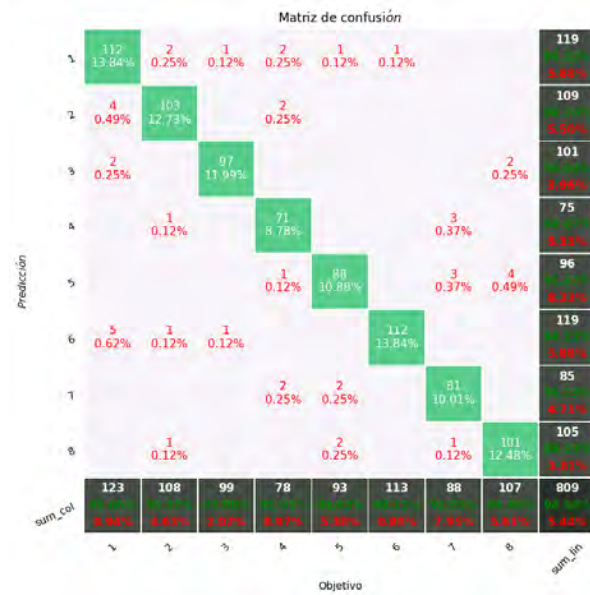
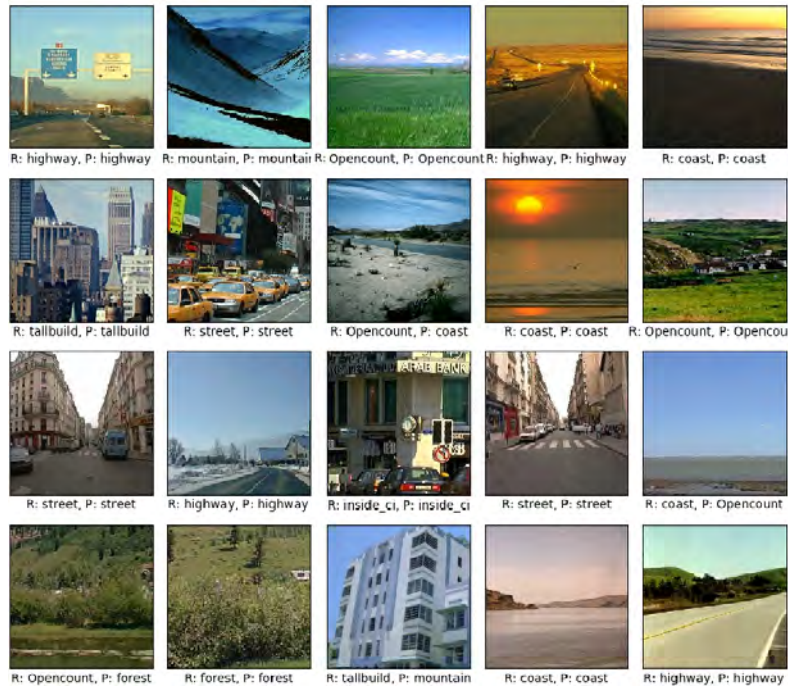


Fig. 6. Matriz de confusión de la red ToniNet (Oliva & Torralba).

- **Resultados con Oliva & Torralba:** Esta es la base de datos relativamente más simple que se utilizó, consta de 1,879 imágenes de entrenamiento y 809 de prueba pertenecientes a 8 clases. Analizando el comportamiento de las redes en la fase de prueba, en la figura 5, podemos observar que las 4 redes aprenden muy bien y alcanzan una buena precisión top-1 en tan solo 150 épocas. Sin embargo, vale la pena mencionar que ToniNet muestra una mejor generalización durante toda la fase de aprendizaje y obtiene el mejor rendimiento con un 94.6% de éxito en la época 250. Con esta base de datos ToniNet supera en 4.2% a AlexNet, en 3.1% a GoogleNet y en 1.8% a ResNet 256.

Para visualizar mejor el desempeño de ToniNet, en la figura 6 se muestra la matriz de confusión con  $C1, C2, \dots, C8$  que corresponden a las clases *Open-country*, *Coast*, *Forest*, *Highway*, *Inside-city*, *Mountain*, *Street* y *Tallbuilding*, respectivamente. En la diagonal principal se observa que la red obtiene un 94.56% de éxito con 765 imágenes correctas de 809.



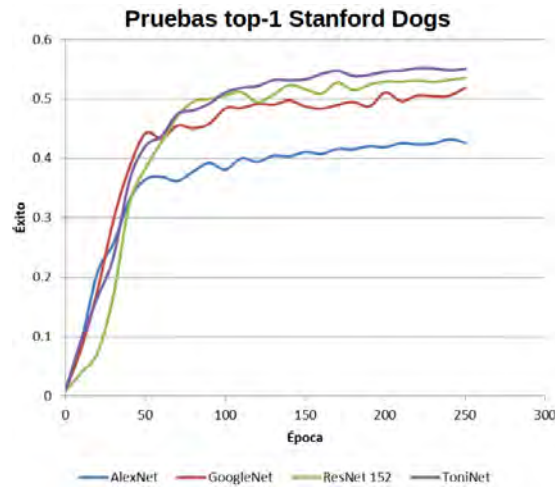
**Fig. 7.** Prueba con Oliva & Torralba, la letra *R* significa que es la clasificación real, la letra *P* significa que es la clasificación dada por la red.

La red tiene la mejor predicción con la clase *Tallbuilding* con el 96.19% seguida de *Forest* con el 96.04%.

La clase con menor éxito de predicción es *Inside.city* con 91.67%, confundiendo con *Street* y *Tallbuilding*. La clase real *Highway* es la más confundida por la red con *Opencountry*, *Coast* y *Street*, también *Opencountry* es confundida con *Coast*, *Forest* y *Mountain*. *Mountain* es la clase que mejor aprende la red. En la figura 7 se muestran algunos de los resultados en la fase de prueba, en donde se puede observar que la red ToniNet confundió una imagen de la clase *coast* por una de la clase *Opencountry*, entre otras que clasificó de forma incorrecta.

- **Resultados con Stanford Dogs:** Esta base de datos es mas complicada que Oliva & Torralba ya que el tamaño de las imágenes es muy variante, contiene 14,358 imágenes de entrenamiento y 6,222 de prueba pertenecientes a 120 clases.

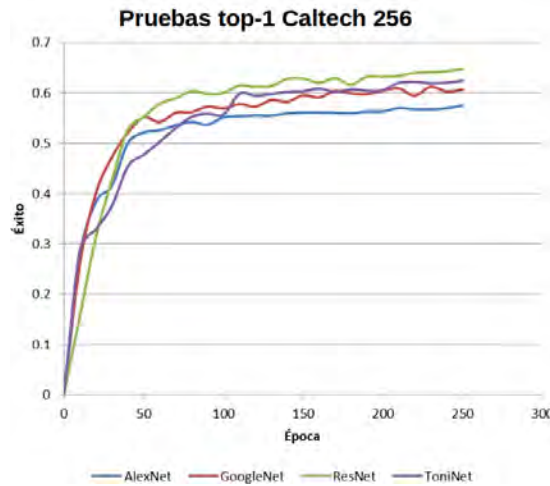
Analizando el comportamiento de las redes en la fase de prueba, en la figura 8, podemos observar que ToniNet tiene el mejor rendimiento con un 55.2% de éxito alcanzado en 250 épocas y supera en 12% a AlexNet, en 3.3% a GoogleNet y en 1.6% a ResNet 256. En esta prueba, ToniNet se llegó a entrenar con 500 épocas logrando un 56.2% como máximo en rendimiento en prueba; sin embargo, el pequeño incremento en el rendimiento es poco significativo comparado al tiempo de entrenamiento.



**Fig. 8.** Resultados de precisión top-1 (Stanford Dogs).

- **Resultados con Caltech 256:** Esta base de datos es más desafiante que Oliva & Torralba y Stanford Dogs, ya que tiene el mayor desequilibrio de imágenes por categoría, contiene 21,314 imágenes de entrenamiento y 9,293 de prueba pertenecientes a 256 clases. Analizando el comportamiento de las redes en la fase de prueba, en la figura 9, podemos observar que ResNet 152 obtuvo el mejor rendimiento con un 64.7% de éxito en 250 épocas, superando

a ToniNet en un 2.2%. Con esta base de datos ToniNet supera en 5.0% a AlexNet, en 1.9% a GoogleNet.



**Fig. 9.** Resultados de precisión top-1 (Caltech 256).

## 6 Conclusiones

La red propuesta dio mejores resultados con el conjunto de imágenes de Oliva & Torralba y StanfordDogs comparándola con AlexNet, GoogleNet y ResNet 152. Fue superada por ResNet 152 con el conjunto de imágenes Caltech 256, pero el tiempo de entrenamiento de nuestra red fue menor y con mayor rendimiento que AlexNet y GoogleNet.

La ventaja de la red con respecto a las comparadas; es que tiene diferentes secciones de extracción de características, con esto necesita menos capas, como por ejemplo: ResNet 152 y GoogleNet. La desventaja principal es que necesita más tiempo de entrenamiento que AlexNet y GoogleNet para lograr mejores resultados pero con menor tiempo que ResNet.

Se ponen las bases para la construcción de redes usando diferentes secciones para la extracción de características. Se pueden añadir más secciones dependiendo del hardware con el que se cuenta. Las redes profundas como ResNet con 152 capas extraen las características solo de una sección, en la red propuesta se pueden extraer características de múltiples secciones.

La red se puede mejorar en la etapa de clasificación, principalmente en las capas totalmente conectadas, también ampliando las secciones.

Como trabajo futuro se planea utilizar la red para la detección de objetos y segmentación de imágenes. También se planea experimentar con más secciones; para trabajar con el color, la forma, la textura, entre otras.

## Referencias

1. Caltech256: Caltech 256 Dataset. [www.vision.caltech.edu/ImageDatasets/Caltech256](http://www.vision.caltech.edu/ImageDatasets/Caltech256) (Mayo 2016)
2. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M., Schmidhuber, J.: Flexible, high performance convolutional neural networks for image classification. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
3. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: international Conference on computer vision & Pattern Recognition (CVPR'05). vol. 1, pp. 886–893. IEEE Computer Society (2005)
4. Fidel López, Andrés Ferreyra, C.A.J.V.A.Z.E.R.: Preprocesamiento de bases de datos de imágenes para mejorar el rendimiento de redes neuronales convolucionales. *Research in Computing Science* **147(7): Robotics and Computer Vision**, 35–45 (2018)
5. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
6. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence* **37(9)**, 1904–1916 (2015)
7. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: Proceedings of the 22nd ACM international conference on Multimedia. pp. 675–678. ACM (2014)
8. Kaiming He, Xiangyu Zhang, S.R.J.S.: Deep residual learning for image recognition. *IEEE Xplore* (2015)
9. Khosla, A., Nityananda, Jayadevaprakash, Yao, B., Fei-Fei, L.: Stanford Dogs Dataset. <http://vision.stanford.edu/aditya86/ImageNetDogs/> (Septiembre 2017)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**, 436–44 (05 2015). <https://doi.org/10.1038/nature14539>
12. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. *International journal of computer vision* **60(2)**, 91–110 (2004)
13. Mairal, J., Koniusz, P., Harchaoui, Z., Schmid, C.: Convolutional kernel networks. In: Advances in neural information processing systems. pp. 2627–2635 (2014)
14. Oliva, Torralba: . <http://cvcl.mit.edu/database.htm> (Mayo 2016)
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014)
16. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
17. Taylor, L., Nitschke, G.: Improving deep learning using generic data augmentation. *CoRR* **abs/1708.06020** (2017), <http://arxiv.org/abs/1708.06020>
18. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using dropout. In: International conference on machine learning. pp. 1058–1066 (2013)
19. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision. pp. 818–833. Springer (2014)

# Preprocesamiento de bases de datos de imágenes para mejorar el rendimiento de redes neuronales convolucionales

Fidel López-Saca, Andrés Ferreyra-Ramírez, Carlos Avilés-Cruz, Juan Villegas-Cortez, Arturo Zúñiga-López, and Eduardo Rodriguez-Martinez

Universidad Autónoma Metropolitana, Azcapotzalco. San Pablo Xalpa No. 180, Col. Reynosa Tamaulipas, CP 02200, México, Ciudad de México.  
fidelosmcc@gmail.com, {fra,caviles,juanvc,azl,erm}@azc.uam.mx

**Resumen** En los últimos años las redes neuronales convolucionales han sido muy populares en el procesamiento de datos a gran escala y muchos trabajos han demostrado que son herramientas muy prometedoras en muchos campos, en especial, en la clasificación de imágenes. Teóricamente, las características de las redes convolucionales pueden mejorar cada vez más con el aumento de las capas de la red; sin embargo, más capas pueden aumentar drásticamente el costo computacional. Además de las características de la red, el tamaño y forma en que se construyen los conjuntos de entrenamiento y prueba, también es un aspecto importante a considerar para mejorar el rendimiento de la red. En este trabajo, proponemos un planteamiento para mejorar el rendimiento de una red neuronal convolucional mediante la división y formación apropiada de los conjuntos de entrenamiento y prueba.

**Keywords:** Redes Neuronales Convolucionales · Tensorflow · Reconocimiento de Imágenes · Procesamiento Digital de Imágenes · Aprendizaje Profundo.

## Image Data Sets Preprocessing to Improve the Performance of Convolutional Neural Networks Implementaion

**Abstract.** Recently, convolutional neural networks have been risen to popularity in tasks such as big data preprocessing and computer vision. Many works have shown that they are a promesing tool in several applications such as digital image classification. Theoretically, a convolutional neural network performance can be increased as the number of layers in its architecture increases, however, more layers can also increase its computational cost. Besides, the network topology, the size and making of the training and testing sets also have a big impact in the network performance. In this work

we proposed a strategy to improve the performance of a convolutional neural network based on the appropriate division of the training and testing sets.

**Keywords.** Convolutional Neural Networks, Tensorflow, Image Recognition, Digital Image Processing, Deep Learning.

## 1. Introducción

Separar los conjuntos de datos en conjuntos de entrenamiento y prueba forma parte importante para el entrenamiento y la evaluación de los modelos de redes neuronales convolucionales (RNCs) [1,2]. Normalmente al dividir el conjunto de datos, la mayoría de los datos se utiliza para formar el conjunto de entrenamiento y una parte menor se emplea para el conjunto de prueba. Los datos son muestreados de forma aleatoria para asegurar que los conjuntos sean representativos de todo el conjunto de datos y para eliminar el sesgo de selección, lo que puede minimizar los efectos de las diferencias entre los datos y comprender mejor las características de la RNC.

Tradicionalmente se selecciona el 70 % de los datos de origen para formar el conjunto de entrenamiento y el 30 % para el conjunto de prueba [3,4]. Sin embargo, aun está abierta la pregunta ¿cuál es la relación ideal para mejorar el rendimiento de una RNC? Esta relación genera un conflicto, con conjuntos de entrenamiento inmensamente grandes se mejora la capacidad de aprendizaje de la RNC, y con conjuntos de prueba grandes se generan intervalos de confianza más precisos. Además, cuando el número de ejemplos de entrenamiento es infinitamente grande e imparcial, los parámetros de la red convergen a uno de los mínimos locales de la función de pérdida empírica a ser minimizada, y cuando este número es finito, la función de pérdida real es diferente a la función de pérdida empírica. En consecuencia, ya que los ejemplos de entrenamiento son parciales, los parámetros de la red convergen a una solución sesgada. Esto se conoce como *sobre-ajuste* o *sobre-entrenamiento* porque los valores de los parámetros se ajustan demasiado bien a la especialidad de los ejemplos de entrenamiento sesgados y nos son óptimos en el sentido de minimizar el error de generalización dado por la función de pérdida [5].

Elegir una relación apropiada para generar los conjuntos de entrenamiento y prueba puede no ser suficiente para obtener mejores rendimientos en RNCs. Muchas de las bases de datos que se utilizan para aplicaciones de clasificación de imágenes contienen categorías con un número de imágenes desigual, por lo que es importante explorar la conveniencia de estandarizar el número de ejemplos por clase (a la clase con el menor número de ejemplos) o utilizar el conjunto completo para generar los conjuntos de entrenamiento y prueba.

En las aplicaciones de clasificación de imágenes con RNCs, se utilizan bases de datos con miles de millones de ejemplos, por lo que, la carga de los datos a la red se convierte en un problema a considerar. Para cargar cada imagen a la red es conveniente generar archivos que contengan la información de las imágenes. Archivos en donde cada imagen es etiquetada automáticamente en

función del nombre de la clase a la que pertenece, lo que permite almacenar datos de imágenes de gran tamaño, incluidos datos que no caben en memoria, y leer lotes de imágenes de manera eficiente durante el entrenamiento de la red.

En este trabajo utilizamos diferentes bases de datos de referencia para entrenar tres RNCs muy conocidas en la literatura, AlexNet [6], GoogleNet [7] y ResNet [8], para probar si estandarizar una base de datos realmente incrementa el porcentaje de rendimiento con las redes neuronales convolucionales, como en otros problemas de clasificación.

Desarrollamos y aplicamos un toolbox para generar los conjuntos de datos de entrenamiento y prueba, los cuales son representados como archivos que contienen la información de las imágenes.

El resto de este documento está organizado de la siguiente manera. En la sección 2 se revisan las redes convolucionales utilizadas. La sección 3 introduce los conjuntos de datos empleados para los experimentos. La sección 4 introduce el toolbox utilizado para generar los conjuntos de entrenamiento y prueba. La sección 5 muestra los experimentos y resultados. Finalmente, la sección 6 discute las conclusiones y el trabajo futuro.

## 2. Redes Neuronales Convolucionales

A continuación se proporciona una breve descripción de las redes que utilizamos en nuestra propuesta.

- **AlexNet:** tiene una profundidad de 8 capas, 5 capas para extracción de características y 3 capas totalmente conectadas. Entre las capas utiliza la función ReLU, la cual reduce el tiempo de aprendizaje, y presenta diferentes variaciones [9]. Esta red implementa la operación de convolución entre la imagen de entrada y un filtro de  $11 \times 11$  en la primera capa, con el objetivo de extraer diferentes características. Las capas de convolución tienen entre 96 y 384 filtros. La red incluyó la normalización local, pooling para extraer los valores más representativos, y softmax para realizar la clasificación de 1,000 clases. Para evitar el sobre entrenamiento, AlexNet implementa el Dropout [10] que principalmente deshabilita un nodo temporalmente así como sus entradas y salidas. La red alcanza la tasa de error de conjunto de pruebas de top-5 de 15,3% con el conjunto de datos ImageNet [11].
- **GoogleNet:** ganadora del concurso ILSVRC2014 en su etapa de clasificación, dejando en segundo lugar a VGGNet [12]. Propone una nueva arquitectura con 22 capas de profundidad, en donde la principal contribución es un módulo llamado Inception que aproxima una CNN dispersa, cuyo resultado es el uso de aproximadamente 12 veces menos parámetros que AlexNet. Los bloques de convolución contienen filtros de tamaño  $1 \times 1$ ,  $3 \times 3$  y  $5 \times 5$ . En conjunto, GoogleNet contiene aproximadamente 100 capas y alcanza una tasa de error sobre el conjunto de pruebas de top-5 de 6,67% usando la base de datos ImageNet.
- **ResNet:** es una red de 152 capas, tiene menos complejidad que GoogleNet pero con mejor precisión con un error de top-5 de 3,57% en el conjunto de



pruebas de ImageNet, tiene diseños con 50 y 101 capas pero la que dio mejor resultado fue la de 152. También tiene bloques internos con filtros entre 64 y 512.

Las redes neuronales convolucionales utilizan variantes del gradiente descendente para el aprendizaje; como son el gradiente descendente estocástico [13], gradiente descendente estocástico con momentos [14], estimación adaptiva del momento (ADAM, adaptive moment estimation) [15], entre otros [13].

### 3. Conjuntos de datos

En este trabajo utilizamos 3 bases de datos diferentes, que tienen como características principales: categorías con un número de imágenes desigual e imágenes con tamaños diferentes. Estas se describen brevemente a continuación:

- **Oliva y Torralba** [16] : Este conjunto de datos consta de 2,688 imágenes a color, pertenecientes a 8 categorías o clases; las imágenes fueron obtenidas de diferentes fuentes: bases de datos comerciales, sitios web y cámaras digitales.
- **ImageNetDogs** [17] : Este conjunto de datos consta de 20,580 imágenes a color, pertenecientes a 120 clases o razas de perros de todo el mundo; las imágenes fueron obtenidas de la base de datos ImageNet [11]. El tamaño por cada imagen es variable.
- **Caltech 256** [18] : Este conjunto de datos consta de 30,607 imágenes a color pertenecientes a 257 categorías, el número mínimo de imágenes en cualquier categoría es de 80.

En la Tabla 1 se muestran las características de las bases de datos.

**Tabla 1.** Bases de datos utilizadas y sus características.

Conjunto de datos	Clases	Dimensiones			No. imágenes		
		Ancho	Alto	Prof	Total	Min x clase	Max x clase
Oliva & Torralba	8	256	256	3	2,688	260	410
ImageNetDogs	120	200 – 500	150 – 500	3	20,580	148	252
Caltech 256	257	300	200	3	30,607	80	827

### 4. Toolbox para generar conjuntos de entrenamiento y prueba

Para generar los conjuntos de entrenamiento y prueba en bases de datos de gran escala, como ImageNet [11], es recomendable generar archivos de tipo *tfrecord* [19]. Archivos que son utilizados para guardar una gran cantidad de imágenes,

que pueden ser de diferentes tamaños, codificadas en arreglos multidimensionales.

Para generar los archivos *tfrecord*, se desarrolló un toolbox, que permite crear los archivos de entrenamiento y prueba de una manera fácil y rápida; además de permitir la generación de parámetros que ayudan al entrenamiento de la red. El toolbox se ejecuta directamente desde la línea de comandos con `python tfpyToolbox.py`, ver Figura 1. Cuando se visualiza la ventana podemos seleccionar la ubicación de las imágenes ya clasificadas, la ruta donde se guardarán los archivos, si se requiere redimensionar la imagen o recortar, el tamaño de la imagen, y el tamaño de los conjuntos de imágenes.

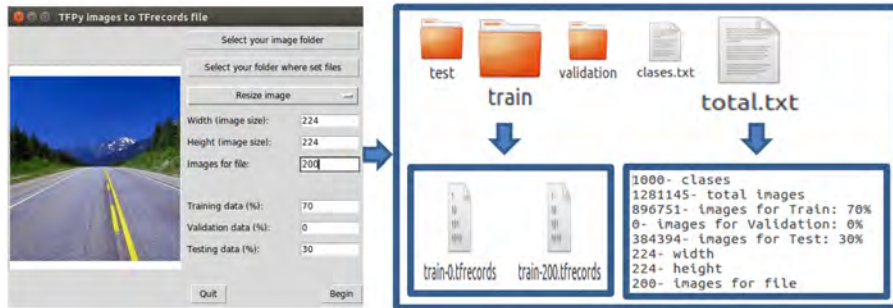


Fig. 1. TFPyToolbox.

La ventaja de este toolbox es que podemos generar conjuntos de entrenamiento y prueba, preparados para ser utilizados en cualquier RNC. El toolbox genera diferentes carpetas con archivos de tipo *tfrecord* y archivos de texto que ayuda en el entrenamiento como son: la descripción de todas las clases, tamaño de la imagen, totales por conjunto de datos. El toolbox separa la base de datos de entrenamiento para utilizarla con diferentes modelos y diferentes parámetros sin necesidad de volver a generar los conjuntos de datos, esta separación se hace por cada clasificación, debido a que las clasificaciones tienen diferentes cantidades de imágenes.

De los archivos *tfrecord* se leen las imágenes y las clases separándolos por lotes y de manera aleatoria, *Tensorflow*<sup>1</sup> tiene la función *shuffle\_batch* para hacerlo, esta función regresa dos lotes, uno de imágenes y otro de clases en nuestro caso será un lote de 32. Cuando se obtienen las imágenes se pueden visualizar algunas para verificar que están de forma correcta y poder continuar con el proceso. En la Figura 2 se muestra de manera general, el proceso de generación y lectura de los archivos.

<sup>1</sup> TensorFlow. <https://www.tensorflow.org>, (Enero 2017) .

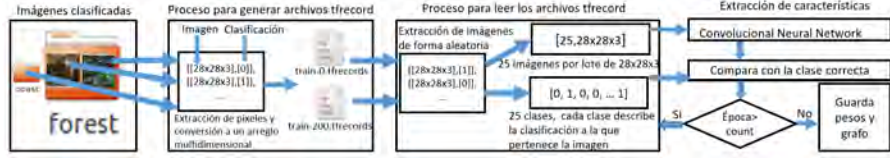


Fig. 2. Escritura y lectura de archivos tfrecord.

## 5. Experimentos y resultados

### 5.1. Parámetros de entrenamiento de las RNCs

Las RNCs fueron entrenadas utilizando ADAM [15] con un tamaño de lote de  $\beta = 32$  imágenes y un decaimiento de pesos (factor de regularización) de  $\lambda = 0,0005$ . Los pesos iniciales en cada una de las capas fueron inicializados con una distribución gaussiana con una media de 0 y una desviación estándar de 0,01. Los umbrales de activación en cada una de las capas fueron inicializados a cero. Iniciamos con una tasa de aprendizaje de  $\mu = 0,001$  la cual se disminuyó en un factor de 10 después de cada 25 épocas, para tener cambios de aprendizaje más específicos en 100 épocas de entrenamiento. Las redes fueron entrenadas en una GPU NVIDIA GeForce GTX TITAN X, con 12 GB de memoria RAM y 3076 núcleos, con sistema operativo Linux Ubuntu 16.04, linux kernel 4.12, Python 2.7, Tensorflow 1.12, NVIDIA CUDA® 8.0, NVIDIA cuDNN v5.1.

### 5.2. Relación entrenamiento/prueba

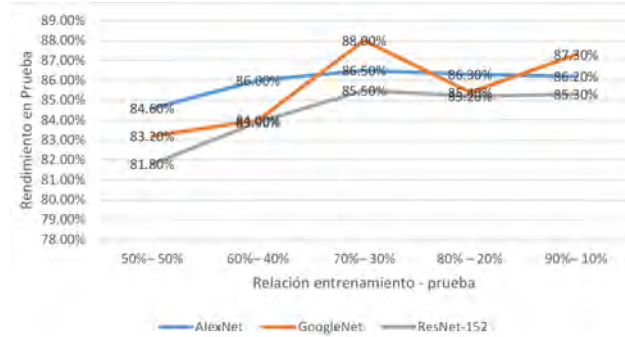
Para evaluar la relación de división de los conjunto de entrenamiento y prueba, elegimos la base de datos Oliva & Torralba utilizando un método de rejilla, variamos la relación desde 50 – 50 hasta 90 – 10 para el porcentaje del conjunto de entrenamiento y prueba respectivamente. Las redes fueron entrenadas desde cero y en cada prueba se registro el rendimiento de la red.

La redes obtuvieron el rendimiento más alto utilizando una relación 70/30 para entrenamiento y prueba, ver Figura 3.

### 5.3. Estandarización de conjuntos de entrenamiento

Para evaluar el efecto de la estandarización o no estandarización de las bases de datos en el rendimiento de las RNCs, planteamos las siguientes pruebas:

1. En la primer prueba, estandarizamos el número de ejemplos por clase de las bases de datos, a la clase con el menor número de imágenes; como se puede ver en la Tabla 2.
2. En la segunda prueba, utilizamos las bases de datos completas, en la Tabla 3 se muestran las características de cada conjunto de datos.



**Fig. 3.** Porcentajes de éxito en entrenamiento y prueba para las distintas redes y el conjunto de datos de Oliva & Torralba.

Para evaluar el rendimiento de las RNCs, las redes se entrenaron desde cero, con cada una de las bases de datos. Para las pruebas se utilizó el método de retención y cada base de datos fue dividida en conjuntos de entrenamiento y prueba. Los conjuntos de entrenamiento fueron formados con el 70 % de las imágenes de cada base de datos y el 30 % restante se utilizó para formar los conjuntos de prueba; la selección de las imágenes se realizó de manera aleatoria. Para ajustar las RNCs a cada uno de los conjunto de entrenamiento, es necesario igualar el número de neuronas de salida al número de clases de cada conjunto.

Los resultados de la prueba 1 se muestran en la Tabla 4, mientras que los resultados de la prueba 2 se muestran en la Tabla 5. Como se puede apreciar en los resultados, para las tres bases de datos y los tres tipos de redes consideradas, los mejores resultados en cuanto a rendimiento, se obtienen cuando se utilizan bases de datos completas. En general GoogleNet tuvo mejores resultados, pero es interesante analizar los dos experimentos. Al realizar la primera prueba con Caltech 256 se dejaron más de 10,000 imágenes fuera del conjunto de datos, al realizar la segunda prueba con la base de datos completa se incrementó el rendimiento para las tres redes. ResNet 152 al tener más capas necesita más cantidad de imágenes para tener mejor rendimiento, al tener mayor profundidad implica mayor cantidad de tiempo en entrenamiento, necesitando 4 veces más que GoogleNet, con las bases de datos utilizadas.

Teóricamente, la prueba 1 intenta quitar el sesgo hacia la clase con el mayor número de ejemplos. De la Tabla 6 se puede ver que el porcentaje de reconocimiento de la clase con mayor número de ejemplos en Caltech 256, *Clutter*, disminuye con la estandarización, pero esto sucede por que se le quitan ejemplos de entrenamiento a la CNN. Por el contrario, el porcentaje de éxito para la mayoría de las clases con menos ejemplos aumenta – las clases *golden-gate-bridge*, *harpichord*, *scorpion-101*, *sunflower-101*, *top-hat* son las que menos ejemplos presentan en Caltech256. Lo que confirma que si se esta disminuyendo el sesgo hacia la clase *Clutter*. Sin embargo, llama la atención que el porcentaje de éxito para la clase con etiqueta *top-hat* disminuye, por lo que se procedió a calcular

**Tabla 2.** Bases de datos separadas en entrenamiento y pruebas, obteniendo la clase que contiene la mínima cantidad de imágenes, así se podrá entrenar con la misma cantidad de imágenes por clase.

Conjunto de datos	Clases	Cantidad de imágenes		
		Entrenamiento	Prueba	Total
Oliva & Torralba	8	1,456	624	2,080
ImageNetDogs	120	12,360	5,400	17,760
Caltech 256	257	14,392	6,168	20,560

**Tabla 3.** Bases de datos separadas en entrenamiento y pruebas, utilizando el 70 % de imágenes por clase para entrenamiento y el 30 % para pruebas.

Conjunto de datos	Clases	Cantidad de imágenes		
		Entrenamiento	Prueba	Total
Oliva & Torralba	8	1,879	809	2,688
ImageNetDogs	120	14,358	6,222	20,580
Caltech 256	257	21,314	9,293	30,607

el porcentaje de mejora promedio para cada una de las clases. Como se observa en la Tabla 7, la prueba 1 incrementa el porcentaje de éxito de 106 clases, mientras que disminuye el de 143 clases. Esta es la razón por la que se obtiene un porcentaje de reconocimiento mejor para la prueba 2.

Como complemento y para mejor análisis de datos, para visualizar las clases que más se confunden, se generaron matrices de confusión por cada prueba, como por ejemplo la Tabla 8 con la prueba de la red GoogleNet y el conjunto de datos Oliva & Torralba con la base de datos completa. Se puede ver que en la diagonal se obtiene la suma de 712 lo que nos da un 88 % de éxito, y que la clase *OpenCountry* es el que más se confunde con la clase *Coast*, el que tiene menor éxito es *Highway* con 70 %. En la Figura 4 se muestran las predicciones obtenidas con algunas de las imágenes utilizadas.

## 6. Conclusiones y trabajo futuro

Separar los datos en archivos para entrenamiento y prueba, permite reutilizar las particiones para diferentes experimentos siempre con los mismo datos, disminuyendo el tiempo de entrenamiento. Dividir un conjunto de datos de imágenes en 70 % para entrenamiento y 30 % para pruebas mejora el rendimiento de las redes neuronales convolucionales. Estandarizar una base de datos a un número mínimo de imágenes (implica tener la misma cantidad de imágenes por clase) baja el

**Tabla 4.** Resultados misma cantidad de imágenes por clase.

CNN	Oliva & Torralba			ImageNetDogs			Caltech 256		
	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5
AlexNet	22	85,4 %	99,7 %	125	30,6 %	60,2 %	146	43,7 %	64,5 %
GoogleNet	26	84,8 %	99,8 %	179	29,6 %	61,9 %	213	42,7 %	65,5 %
ResNet 152	105	84,3 %	99,4 %	769	13,5 %	38,1 %	887	23,2 %	43,3 %

**Tabla 5.** Resultados utilizando las bases de datos completas.

CNN	Oliva & Torralba			ImageNetDogs			Caltech 256		
	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5	Minutos	Top-1	Top-5
AlexNet	23	86,5 %	99,5 %	122	33,4 %	64,4 %	147	49,9 %	70,1 %
GoogleNet	27	88,0 %	100 %	175	30,5 %	64,5 %	215	50,8 %	71,8 %
ResNet 152	106	85,5 %	99,5 %	760	14,7 %	41,0 %	890	32,9 %	54,5 %

**Tabla 6.** Comparación entre la prueba uno y la prueba dos con las clases con mayor cantidad de imágenes y menor cantidad de imágenes del conjunto de datos Caltech 256, con las clases  $C1, C2, \dots, C6$  que corresponden a *Clutter*, *golden-gate-bridge*, *harpsichord*, *scorpion-101*, *sunflower-101*, *top-hat*.

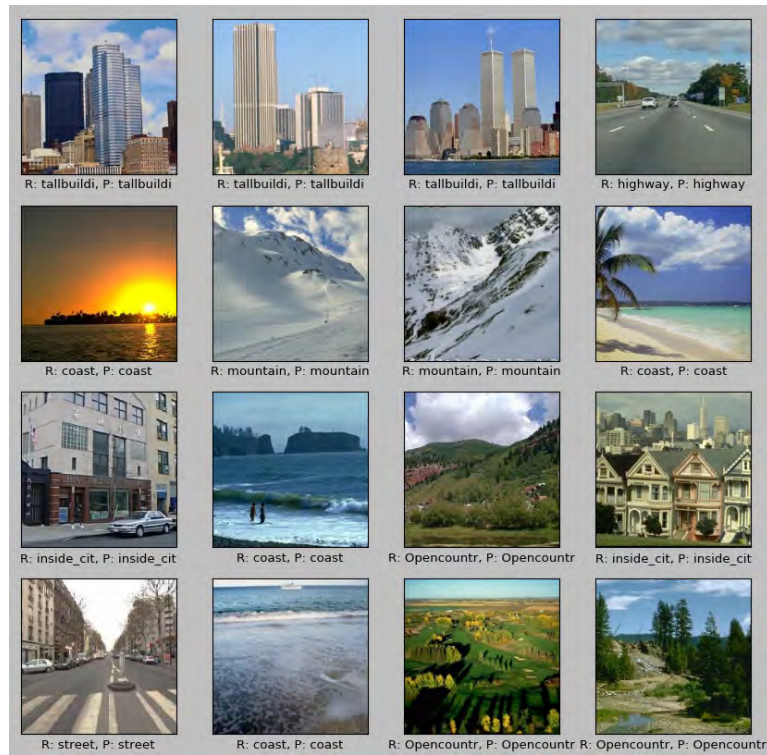
Clase	Imágenes	Prueba 1	Prueba 2
C1	827	0,25 %	0,61 %
C2	80	0,58 %	0,54 %
C3	80	0,75 %	0,41 %
C4	80	0,54 %	0,41 %
C5	80	0,83 %	0,79 %
C6	80	0,37 %	0,54 %

**Tabla 7.** Porcentaje de mejora promedio para cada una de las clases estandarizadas en Caltech-256.

	Núm. clases	Porcentaje
Clases con mayor rendimiento	106	0,045
Clases con menor rendimiento	143	0,065
Clases sin cambiar	8	0,0

**Tabla 8.** Matriz de confusión con las clases  $C1, C2, \dots, C8$  que corresponden a *Open-country*, *Coast*, *Forest*, *Highway*, *Inside-city*, *Mountain*, *Street*, *Tallbuilding*, que contiene el conjunto de datos de Oliva & Torralba, tiene un éxito en pruebas de 88 % con 712 imágenes correctas en la predicción de la red GoogleNet, también se puede ver que la clase que más se confunde es *Open-country* con la clase *Coast*.

	C1	C2	C3	C4	C5	C6	C7	C8	Total
C1	106	11	0	3	0	3	0	0	123
C2	3	96	1	2	0	6	0	0	108
C3	3	0	89	0	0	7	0	0	99
C4	7	9	0	55	1	1	4	1	78
C5	0	1	0	0	89	0	2	1	93
C6	3	2	2	0	0	106	0	0	113
C7	0	0	1	2	6	2	76	1	88
C8	1	3	0	0	5	3	0	95	107
Total	123	122	93	62	101	128	82	98	809



**Fig. 4.** Prueba de la red AlexNet, la letra  $R$  significa que es la clasificación real, la letra  $P$  significa que es la predicción de la red.

rendimiento, debido a que se dejan de utilizar imágenes para el entrenamiento, sin embargo si se logra reducir el sesgo hacia la clase dominante. Realizar la separación por clase (implica tener la base de datos completa), aumenta el porcentaje de éxito para los conjuntos de datos utilizados en este trabajo. A mayor cantidad de imágenes se obtienen mejores resultados para las redes con mayor profundidad, pero con mayor tiempo en entrenamiento, implica un aumento en el costo computacional. Sin embargo, cabe destacar que es posible que en bases más grandes se puede ajustar mejor y aumentar el rendimiento normalizando las clases. El entrenamiento de una RNC es la base para el aprendizaje, analizar los datos de entrenamiento y prueba puede ayudar a reducir los tiempos y a mejorar el aprendizaje. En este trabajo se ponen las bases para crear las propias bases de datos de imágenes, de entrenamiento y prueba utilizando archivos *tfrecord* para utilizarlas en redes neuronales convolucionales.

Como trabajo futuro se plantea experimentar con el pre-procesado de la imagen antes del ingreso a la red, modificando características como el enfoque, la claridad, recorte (*crop*), relleno (*padding*), entre otros, para simular la riqueza de imágenes. Implica mayor costo computacional, pero se espera un mejor rendimiento. Otra línea implica probar diferentes técnicas de muestreo para sobre-muestrear las clases no dominantes sin llegar a producir sobre-entrenamiento.

## Referencias

1. R. Tibshirani, G. James, D. Witten, and T. Hastie, *An introduction to statistical learning-with applications in R*, p. 176. New York, NY: Springer, 2013.
2. S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, p. 709. Malaysia; Pearson Education Limited, 3rd ed., 2009.
3. J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, ch. 7, pp. 219–259. Springer series in statistics New York, 2nd ed., 2009.
4. B. Lei, G. Xu, M. Feng, F. van der Heijden, Y. Zou, D. de Ridder, and D. M. Tax, *Classification, parameter estimation and state estimation: an engineering approach using MATLAB*, ch. 5, pp. 139–182. John Wiley & Sons, 2nd ed., 2017.
5. S.-i. Amari, N. Murata, K.-R. Muller, M. Finke, and H. H. Yang, “Asymptotic statistical theory of overtraining and cross-validation,” *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 985–996, 1997.
6. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.
7. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CoRR*, vol. abs/1409.4842, 2014.
8. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015.
9. D. Mishkin, N. Sergievskiy, and J. Matas, “Systematic evaluation of CNN advances on the imagenet,” *CoRR*, vol. abs/1606.02228, 2016.
10. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.



11. P. U. Stanford Vision Lab, Stanford University, “Imagenet.” <http://imagenet.org/>, Octubre 2017.
12. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014.
13. S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.04747, 2016.
14. I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” *Proceedings of the 30th International Conference on Machine Learning*, 2013.
15. J. L. B. Diederik P. Kingma, “Adam: a method for stochastic optimization,” *arXiv:1412.6980v9*, 2017.
16. Oliva and Torralba, “Computational visual cognition laboratory.” <http://cvcl.mit.edu/database.htm>, Mayo 2016.
17. A. Khosla, Nityananda, Jayadevaprakash, B. Yao, and L. Fei-Fei, “Stanford Dogs Dataset.” <http://vision.stanford.edu/aditya86/ImageNetDogs/>, Septiembre 2017.
18. Caltech256, “Caltech 256 dataset.” [www.vision.caltech.edu/ImageDatasets/Caltech256](http://www.vision.caltech.edu/ImageDatasets/Caltech256), Mayo 2016.
19. Tensorflow, “Importing data.” [https://www.tensorflow.org/programmers\\_guide/datasets](https://www.tensorflow.org/programmers_guide/datasets), Enero 2018.

# PROTOTIPO FUNCIONAL PARA CLASIFICACIÓN DE IMÁGENES CON SALIDA DE AUDIO EN UN SISTEMA EMBEBIDO CON RED NEURONAL CONVOLUCIONAL

***Fidel López Saca***

Universidad Autónoma Metropolitana Azcapotzalco

*fidelosmcc@gmail.com*

***Andrés Ferreyra Ramírez***

Universidad Autónoma Metropolitana Azcapotzalco

*fra@correo.azc.uam.mx*

***Carlos Avilés Cruz***

Universidad Autónoma Metropolitana Azcapotzalco

*caviles@correo.azc.uam.mx*

## Resumen

En los últimos años, las redes neuronales convolucionales, han tenido una gran popularidad en aplicaciones de clasificación de imágenes, principalmente porque superan en rendimiento a los algoritmos tradicionales. Sin embargo, su alto costo computacional complica su implementación en sistemas embebidos con pocos recursos como las Raspberry Pi 3. Para superar este problema, se puede hacer uso del “Neural Compute Stick”, un dispositivo desarrollado recientemente, que integra una GPU en la que se puede cargar una red neuronal convolucional pre-entrenada. En este artículo se presenta un prototipo basado en la Raspberry Pi 3, que realiza clasificación de imágenes con reproducción de audio. La clasificación se realiza con la red GoogleNet, la cual es entrenada fuera de línea, implementada en un NCS e integrada a la tarjeta Raspberry Pi 3. En el sistema propuesto, la imagen que ingresa a través de una cámara web, es clasificada y etiquetada con la red convolucional y finalmente la etiqueta es traducida en audio por el sistema embebido para describir el objeto encontrado en la imagen.

**Palabras Claves:** Movidius stick, Raspberry, Red neuronal convolucional, TensorFlow, Visión por computador.

## **Abstract**

*In recent years, convolutional neural networks (CNN) have become very popular in image classification applications, mainly because they outperform traditional algorithms in performance. However, its high computational cost complicates its implementation in embedded systems with few resources such as Raspberry Pi 3. To overcome this problem, a "Neural Compute Stick" (NCS) can be used, which integrates a GPU. In the NCS can be loaded a pre-trained convolutional neural network. This article presents a prototype based on a Raspberry Pi 3, which performs image classification with audio reproduction. The classification is done through a GoogleNet net, which is trained offline, implemented in the NCS and integrated with the Raspberry Pi 3 card. In the proposed system, the image that enters through a webcam is classified and tagged with the CNN. Finally, the tag is translated into an audio file to be heard.*

**Keywords:** Movidius stick, Raspberry, Convolutional neural network, TensorFlow, Computer vision.

## **1. Introducción**

El aprendizaje profundo es un subcampo específico del aprendizaje automático, un enfoque que utiliza arquitecturas compuestas de múltiples capas de transformaciones no lineales, para aprender representaciones a partir de datos y que pone énfasis en el aprendizaje de capas sucesivas de representaciones cada vez más significativas. Una de las arquitecturas más populares utilizadas hoy en día son las redes neuronales convolucionales (CNN, por sus siglas en inglés), ya que cada una de sus capas, modela un campo receptivo de la corteza visual primaria del cerebro biológico; lo que las hace muy efectivas en tareas de visión artificial.

Dado que las CNN emulan la visión humana, tienen una alta precisión para tareas de reconocimiento de imágenes y han demostrado un rendimiento casi

humano en muchas tareas de visión artificial del mundo real, incluyendo la clasificación de imágenes [Ciresan and Meier., 2011] [Krizhevsky et al., 2012], el reconocimiento [Zhang et al., 2015] [Mollahosseini et al., 2016] y la detección de objetos [Sermanet et al., 2014] [Girshick et al., 2014], el diagnóstico por imágenes médicas [Prasoon et al., 2013], la segmentación de escenas y el etiquetado [Karpathy et al., 2015], etc.

El entrenamiento de una CNN compleja representa un alto costo computacional, por lo que, la mayoría de modelos están entrenados e implementados con plataformas de software, utilizando potentes unidades de procesamiento gráfico (GPU, por sus siglas en inglés) [Vasilache et al., 2014] [Strigl et al., 2010] como motores de cálculo, que tienen numerosas unidades de ejecución y gran ancho de banda de memoria para obtener un alto rendimiento computacional.

En aplicaciones de reconocimiento en tiempo real, las redes pequeñas (con pocas capas) han sido implementadas en dispositivos móviles con pocos recursos de memoria y procesamiento, por ejemplo: MobileNet [Howard et al., 2017] entrenada con ImageNet [Stanford, 2017] para la clasificación de 1000 objetos diferentes, se ha implementado en celulares con sistema operativo Android para detectar la retinopatía diabética, entrenada y probada con más de 16,000 imágenes preprocesadas para la detección de este problema [Suriyal, 2018]. Sin embargo, redes muy grandes (con muchas capas) no pueden ser implementadas en dispositivos móviles, ni en sistemas embebidos de bajo costo tales como: Arduino, Beagle y Raspberry, ya que sus procesadores de propósito general no han sido optimizados para la implementación de estas redes.

Las CNN han sido implementadas en sistemas de visión embebidos para aplicaciones tales como: reconocimiento de escritura a mano [LeCun et al., 1998], detección de rostros [Sankaradas et al., 2009], reconocimiento de objetos [Farabet et al., 2010], y determinación de escenas [Peemen et al., 2013]. En estos sistemas de visión embebidos, el entrenamiento de la CNN se realizó fuera de línea, y el modelo pre-entrenado se ejecutó en tiempo real. Utilizaron procesadores demasiado lentos lo que demandaba la aceleración de la CNN para la ejecución en tiempo real, la cual solo se podía lograr con la integración de GPU en los

sistemas embebidos; sin embargo, la solución no era muy adecuada ya que las GPU consumían muy alta potencia.

A principios del 2014, empezaron a surgir supercomputadoras móviles para el desarrollo de sistemas embebidos las cuales aprovechan los recursos de cómputo de procesadores de primera generación (con diferentes arquitecturas y GPU) que posibilitan el desarrollo de una nueva generación de aplicaciones que emplean la visualización por computadora, el procesamiento de imágenes y el procesamiento de datos en tiempo real, para los sectores de robótica, medicina, aeroespacial y fabricación de automóviles. Estas plataformas año con año duplican su rendimiento y reducen el consumo de potencia, lo que las hace muy atractivas para entrenar e implementar CNN embebidas; sin embargo, tienen un problema, son relativamente costosas y de difícil acceso.

Los problemas de implementación de CNN en sistemas embebidos de bajo costo como la Raspberry Pi 3, pueden ser resueltos con el uso de un Neural Compute Stick (NCS) [Xu, 2017], un dispositivo con conexión USB de bajo consumo de potencia y costo, que integra una GPU que soporta la carga de redes diseñadas y entrenadas en plataformas de aprendizaje profundo como Caffe [Caffe, 2018] y TensorFlow [TensorFlow, 2018]; éste dispositivo soporta redes tales como: AlexNet [Krizhevsky et al., 2012], GoogleNet [Szegedy et al., 2015] y ResNet [He et al., 2016].

La implementación de CNN en sistemas embebidos de bajo costo, puede acelerar el camino de la computación embebida hacia el futuro, permitiendo el desarrollo de aplicaciones en donde las máquinas interactúen y se adapten a sus entornos en tiempo real. En este artículo se presenta un prototipo basado en la Raspberry Pi 3, que realiza clasificación de imágenes con reproducción de audio. La clasificación es realizada por la CNN GoogleNet, la cual es entrenada fuera de línea, implementada en un NCS e integrada a la tarjeta Raspberry Pi 3. El trabajo está organizado de la siguiente manera: la sección 2 contiene la descripción de la metodología utilizada, describiendo partes importantes del proceso, el hardware y el software utilizado, la sección 3 muestra los resultados del entrenamiento, prueba y la implementación de la red en la Raspberry Pi 3, en la sección 4 se

discuten los resultados, y finalmente en la sección 5 se muestran las conclusiones de este proyecto.

## 2. Métodos

El prototipo está dividido en dos etapas: la etapa de entrenamiento y evaluación de la CNN y la etapa de implementación de la red en el sistema embebido Raspberry Pi 3; ver figura 1. En esta sección se describe: cada uno de los dispositivos principales del prototipo, el entrenamiento y la evaluación de la CNN, la implementación de la CNN en el NCS, y el conjunto de datos utilizado.

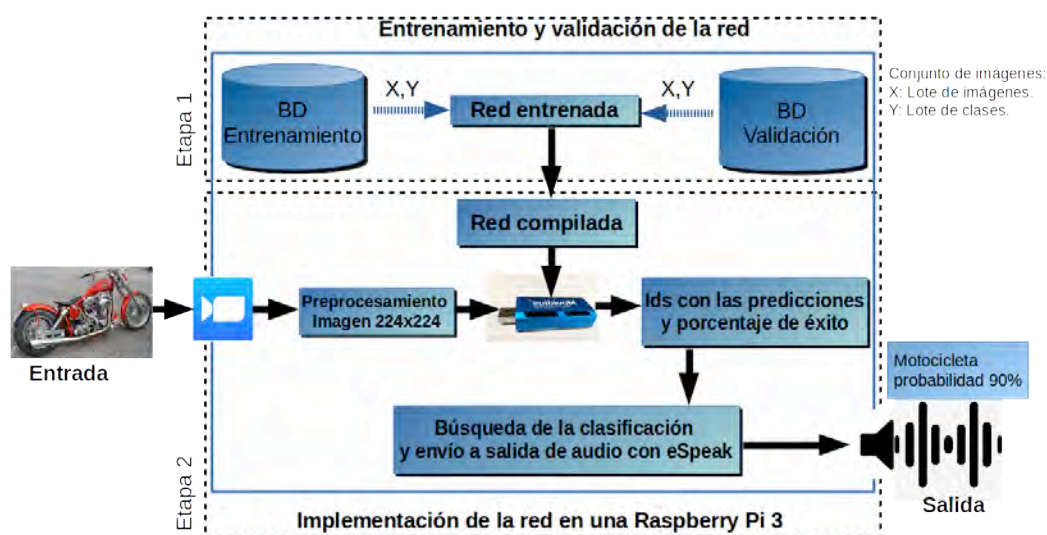


Figura 1 Metodología general.

### A) Raspberry Pi 3

La Raspberry Pi 3 [Raspberry Pi, 2018], es una poderosa mini computadora basada en un procesador ARMv8 Cortex A53, un procesador de cuatro núcleos a 1.2 GHz de 64 bits. Está equipada con: 1 GB de memoria RAM, 802.11n Wireless LAN (10/100Mbps de velocidad), Bluetooth 4.1 de baja energía (BLE), 4 puertos USB, 40 pines GPIO (General Purpose Input Output), puerto HDMI completo, Interfaz de cámara (CSI), Interfaz de pantalla (DSI), ranura para tarjeta Micro SD, salida estéreo y un procesador gráfico VideoCore IV 3D, ver figura 2. Su precio asequible, compatibilidad con otros dispositivos, alta flexibilidad, rendimiento

adecuado, recursos disponible y una amplia comunidad de usuarios; hacen de esta plataforma, la elección ideal para desarrollar aplicaciones embebidas basadas en aprendizaje automático.

## **B) Neural Compute Stick (NCS)**

El Intel® Movidius™ Neural Compute Stick, es un pequeño dispositivo USB 3.0 de bajo costo diseñado para implementar redes de aprendizaje profundo, en especial CNN, en aplicaciones de baja potencia que requieren inferencia en tiempo real. Internamente contiene un procesador Movidius™ Myriad™ 2, una unidad de procesamiento de visión (VPU, por sus siglas en inglés) que ofrece un alto rendimiento en entornos con energía limitada y su arquitectura no depende de una conexión a Internet. Está diseñado para trabajar en un rango de temperatura de 0 a 40 °C y sus dimensiones son de 72.5mm x 14mm, ver figura 2. En [Xu, 2017] se muestran comparaciones de su capacidad de procesamiento.

El NCS tiene un toolkit “Intel® Movidius™ NCS Quick Start” [NCS Quick Star, 2018], y ejemplos “Neural Compute Application Zoo (NC App Zoo)” [Neural, 2018] con Caffe [Caffe, 2018] y TensorFlow [TensorFlow, 2018].

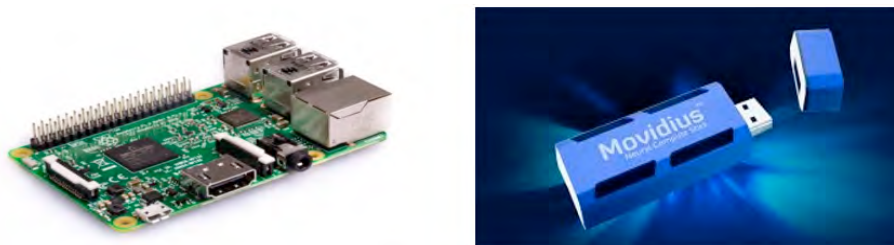


Figura 2 Dispositivos principales del prototipo: Raspberry Pi 3 (izquierda) y el Neural Compute Stick (derecha).

## **C) Red neuronal convolucional: entrenamiento y prueba**

La CNN adoptada para el sistema propuesto es GoogleNet [Szegedy et al., 2015], un modelo de 144 capas que se puede dividir en diferentes secciones básicas: el tallo (stem), los módulos de inicio (inception modules), clasificadores auxiliares, y el clasificador de salida. El tallo está formado por una secuencia de operaciones de convolución, agrupación (pooling) y operaciones de normalización

de respuesta local. Cada uno de los nueve módulos de inicio, está formado por un conjunto de convoluciones y agrupaciones a diferentes escalas, cada una hecha en paralelo y posteriormente concatenadas. Los clasificadores auxiliares se ramifican desde la red principal, y tienen como objetivo amplificar la señal de gradiente a través de la red, intentando mejorar las representaciones anteriores de los datos. Finalmente el clasificador de salida, realiza una operación de agrupación promedio (average pooling) seguida de una activación "softmax" en una capa totalmente conectada.

La CNN fue entrenada utilizando descenso de gradiente estocástico con un tamaño de lote de 32, momento de 0.9 y una disminución de pesos (factor de regularización) de 0.00004. Los pesos iniciales en cada una de las capas de convolución y completamente conectas, fueron inicializados utilizando el algoritmo de inicialización Xavier; en el cual, los pesos son aleatoriamente inicializados con una media cero y una varianza que depende de la cantidad de conexiones entrantes y salientes de la capa. Los umbrales de activación en cada una de las capas fueron inicializados a cero. El número de épocas de entrenamiento se fijó a 650. Se inició con una tasa de aprendizaje de 0.001 la cual se disminuyó en un factor de 0.98 después de cada 2 épocas.

Para evaluar el rendimiento de la CNN, la red se entrenó desde cero, y la base de datos utilizada fue dividida en conjunto de entrenamiento y prueba. El conjunto de entrenamiento fue formado con el 70% de las imágenes de la base de datos y el 30% restante se utilizó para formar el conjunto de prueba; estos conjuntos fueron guardados en archivos tfrecord [TensorFlow, 2018]. La CNN fue entrenada y validada en una Workstation con 16 GB de memoria RAM, utilizando una GPU NVIDIA GeForce GTX TITAN X con 3072 núcleos y 12 GB de memoria, utilizando el marco de aprendizaje profundo TensorFlow [TensorFlow, 2018].

#### **D) Implementación de la red en la Raspberry Pi 3**

La Raspberry Pi 3 a través de una cámara web, captura la imagen a analizar en formato RGB, la imagen es redimensionada a un tamaño de 224x224x3 ajustándola al tamaño requerido por la capa de entrada de la CNN. La imagen



redimensionada es enviada al NCS para determinar la categoría de los objetos que contiene. Finalmente la Raspberry Pi 3, genera y reproduce una salida de audio que describe la clase a la que pertenece la imagen visualizada. El software eSpeak [eSpeak, 2018], fue instalado en el sistema operativo y utilizado para convertir el texto que describe la clase en audio, el cual es reproducido por la bocina. El software incluye diferentes voces y permite alterar sus características.

El NCS es el dispositivo en donde se carga la CNN preentrenada para la aplicación. El NCS recibe la imagen de la Raspberry Pi 3 y realiza la clasificación del objeto preponderante que contiene la imagen. La predicción es regresada a la Raspberry Pi 3 para que ésta genere y reproduzca la salida de audio que describe la clase del objeto visualizado.

En la figura 3, se muestra la integración de los elementos principales del prototipo propuesto, la Raspberry Pi 3 y el “Neural Compute Stick”. La programación de la Raspberry Pi 3 se hizo en Python 2.7, usando la librería OpenCV 3.4. La Raspberry captura 18 fotogramas cada 3 segundos, tiempo en el cual, el NCS realiza 18 clasificaciones de las imágenes instantáneas. Si durante los 3 segundos, la salida del NCS es la misma clase, la Raspberry traduce en audio la etiqueta de la clase para describir el objeto encontrado en la imagen y guarda en disco una copia de la imagen junto con su etiqueta.



Figura 3 Prototipo funcional para clasificación de objetos en imágenes con salida de audio en un sistema embebido con red neuronal convolucional.

A continuación se proporciona un procedimiento simple, basado en TensorFlow, para cargar la CNN preentrenada en el NCS; cabe aclarar que la CNN es diseñada y preentrenada en un servidor utilizando herramientas apropiadas para ese fin.

1. En primer lugar, se entrena la CNN diseñada utilizando una base de datos para la aplicación deseada. El entrenamiento se realiza bajo el ambiente de trabajo, TensorFlow y Python. Al final del entrenamiento, TensorFlow genera y guarda tres archivos: `.index`, `.meta` y `.data`, los cuales contienen toda la información del entrenamiento.
2. En segundo lugar, se realiza una transferencia de aprendizaje de la CNN entrenada, en donde se remueve de la red: el algoritmo de aprendizaje y las últimas dos capas "Dropout" y "Classification". Con esto, la capa Softmax es declarada como la capa de salida de la red, generando como variable de salida `varOutput = softmax()` la cual dará las predicciones de las distintas clases. Al final de este paso, se generan y guardan nuevamente los tres archivos con las extensiones mencionadas en el punto anterior.
3. Los tres archivos de inferencia generados en el punto 2, son utilizados para generar un archivo con extensión `*.graph` a través del compilador del SDK (kit de desarrollo de software) del NCS. Esta tarea es realizada con el comando siguiente: `mvnNCCompile MiRed_inference.meta -s 12 -in variable_entrada -on variable_salida -o MiRed_inference.graph`. Para más detalles sobre la compilación consultar [NC SDK, 2018].
4. Una vez generado el archivo `*.graph` que contiene la CNN entrenada, se codifica para que sea cargado en el NCS; la Raspberry Pi 3 recibe las clasificaciones generadas por el NCS para cada imagen de consulta.

Los primeros tres puntos del procedimiento anterior se realizan en el servidor, el punto cuatro se implementa en la Raspberry Pi 3.

## E) Base de datos

En este trabajo, se utilizó el conjunto de datos “Visual Object Classes Challenge 2012 (VOC2012)” [Everingham, 2018], la cual consta de 11,540 imágenes a color de alta resolución pertenecientes a 20 clases. Cada clase contiene de 303 a 4,087 imágenes y la mayoría de las categorías tienen alrededor de 550 imágenes. Se seleccionaron cinco clases (“carro”, “gato”, “perro”, “motocicleta” y “persona”), para formar un subconjunto con 8,140 imágenes; donde la clase con el menor número de imágenes es “motocicleta” con 526, y la clase con el mayor número de imágenes es “persona” con 4,087. Para no lidiar con el desbalance de las clases del subconjunto de imágenes, se estandarizó el número de imágenes por clase, a la clase con el menor número de imágenes, para formar un conjunto de 2,630 imágenes; este es el conjunto base para el entrenamiento y prueba de la CNN. La estandarización se realizó de manera aleatoria.

## F) Requisitos del prototipo

En la tabla 1 se resumen las características del Hardware y el Software utilizado para el desarrollo del prototipo propuesto.

Tabla 1 Características de Hardware y Software.

<b>Hardware</b>	Entrenamiento: <ul style="list-style-type: none"><li>• GPU NVIDIA GeForce GTX TITAN X, con 12 GB de memoria RAM y 3076 núcleos.</li><li>• Workstation Dell T7600, Procesador Intel(R) Xeon(R), 16 Gb de Memoria RAM, 1 disco duro de 500 GB.</li></ul>
	Implementación: <ul style="list-style-type: none"><li>• Neural Compute Stick: USB 3, 4 GB LPDDR3, Precisión FP16.</li><li>• Raspberry Pi 3 Model B: CPU Quad Core 1.2 GHz, Broadcom, BCM2837 de 64bits, 1 GB de RAM, Chip BCM43438 wireless LAN y Bluetooth, GPIO de 40 pines, 32 GB de memoria Micro SD.</li><li>• Cámara Web.</li><li>• Bocina con entrada plug 3.5 mm.</li></ul>

Tabla 1 Características de Hardware y Software (continuación).

<b>Software</b>	<p>Entrenamiento:</p> <ul style="list-style-type: none"> <li>• Sistema operativo Linux Ubuntu 16.04, kernel 4.12.</li> <li>• Python 2.7.</li> <li>• TensorFlow v1.4.</li> <li>• NVIDIA CUDA® 8.0.</li> <li>• NVIDIA cuDNN v5.1.</li> </ul>
	<p>Implementación:</p> <ul style="list-style-type: none"> <li>• Sistema operativo Raspbian Stretch.</li> <li>• Neural Compute SDK.</li> <li>• OpenCV 3.4.</li> <li>• eSpeak text to speech v18.</li> </ul>

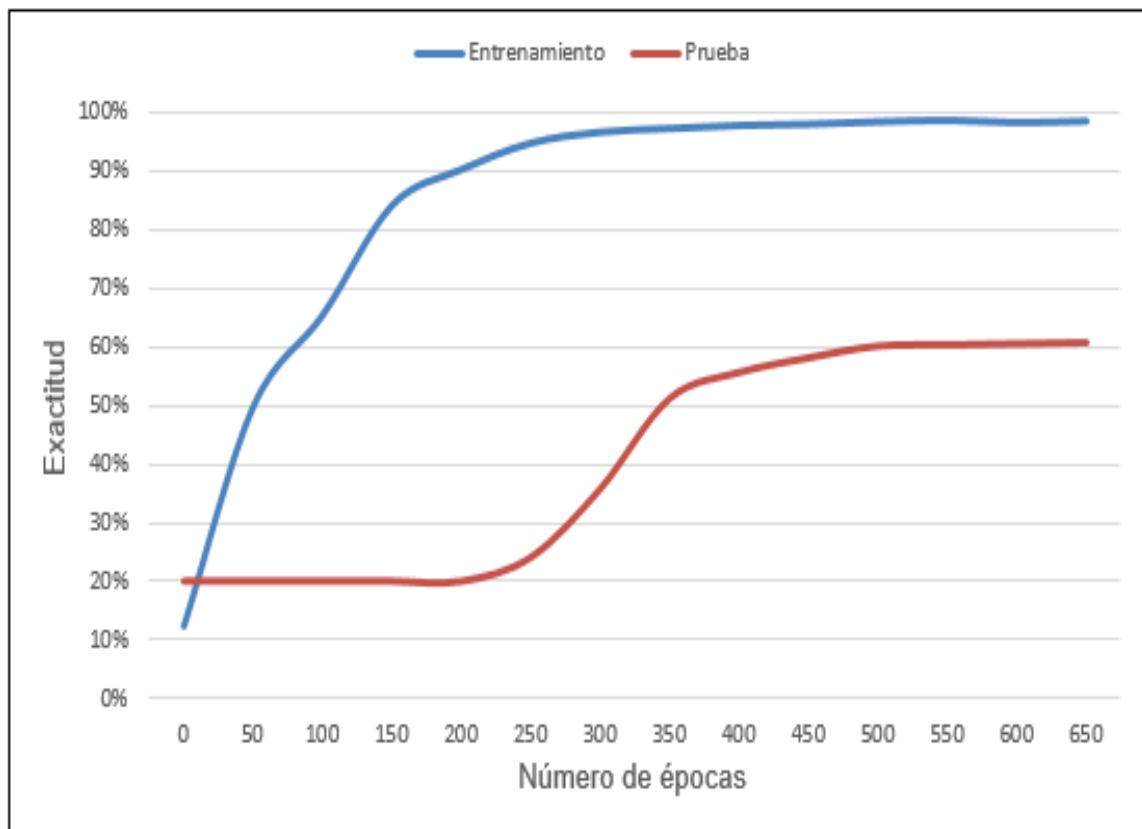


Figura 4 Gráficas de exactitud de la CNN entrenada en 650 épocas.

### 3. Resultados

#### A) Entrenamiento y prueba de la CNN

En la figura 4 se muestran las gráficas de exactitud de entrenamiento y prueba obtenidas para la red GoogleNet después de 650 épocas de entrenamiento. En entrenamiento, la CNN llega a una exactitud del 98.75% en las 650 épocas, mientras que en la fase de prueba la red logra un rendimiento máximo de 60.13%. Obsérvese que después de la época 450, la exactitud de entrenamiento se incrementa mientras que la exactitud de prueba permanece constante, lo que indica, que la red quedó ajustada a características muy específicas de los datos de entrenamiento; es decir, la red sufre de sobreajuste.

Para visualizar mejor el rendimiento de la CNN en la fase de prueba, en la figura 5, se muestra una matriz de confusión, en la cual cada fila representa el número de predicciones de cada clase, mientras que cada columna representa los ejemplos de una clase real. Las celdas diagonales corresponden a los ejemplos que están clasificados correctamente. Las celdas fuera de la diagonal corresponden a ejemplos incorrectamente clasificados. En cada celda se muestran tanto el número de ejemplos como el porcentaje del número total de ejemplos. La columna en el extremo derecho de la gráfica muestra los porcentajes de todos los ejemplos que se predice que pertenecen a cada clase que están clasificados correcta e incorrectamente. La fila en la parte inferior de la gráfica muestra los porcentajes de todos los ejemplos que pertenecen a cada clase que están clasificados correcta e incorrectamente. Finalmente, la celda en la parte inferior derecha de la gráfica muestra la precisión general. Las clases con el mejor rendimiento fueron “gato” (2) y “motocicleta” (4) con el 67.72% y 67.09% respectivamente; mientras que la clase con el peor rendimiento fue “perro” con el 46.20%. La red tiene problemas para distinguir entre “carro” y “motocicleta”, y entre “gato” y “perro”. La red también confunde “persona” con “carro” o “motocicleta”, esto debido a que una gran cantidad de imágenes de la clase “persona” contienen carros o motocicletas.

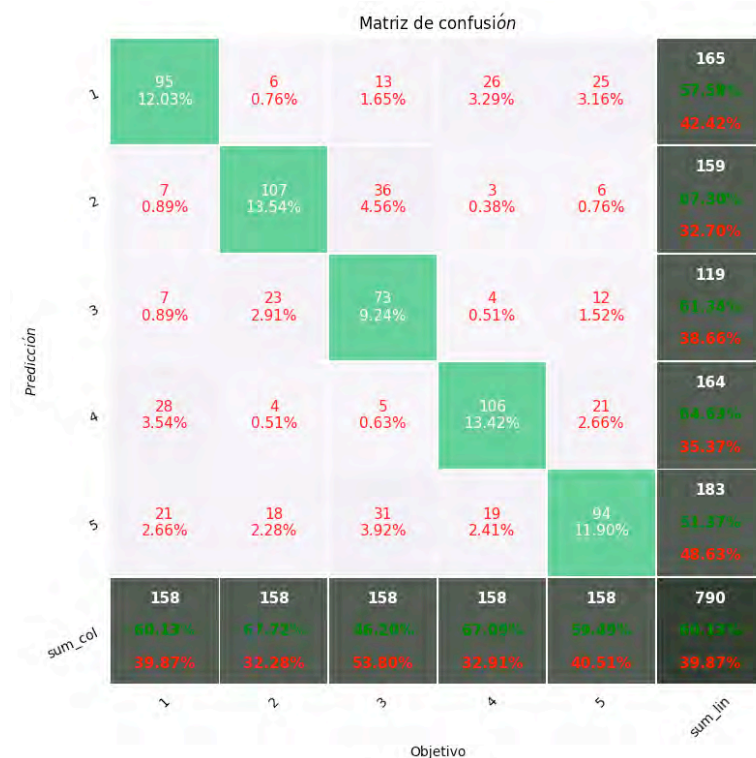


Figura 5 Matriz de confusión para la fase de prueba de la CNN. Las etiquetas 1, 2, 3, 4 y 5, representan a las clases “carro”, “gato”, “perro”, “motocicleta” y “persona”, respectivamente.

## B) Implementación de la red en la Raspberry Pi 3

Las pruebas se llevaron a cabo en el estacionamiento de la universidad para detectar autos, personas y motocicletas; para detectar perros y gatos las pruebas se realizaron en diferentes hogares. Se experimentó con 10 pruebas por clase a una distancia de entre un metro y tres metros para clasificar a las personas. Para los carros y las motocicletas, la distancia fue de tres a cinco metros como máximo. En la figura 6 se muestran los resultados de las predicciones de las 5 clases, con imágenes reales, utilizando el prototipo. El prototipo logró predecir con una exactitud del 100% a las clases “carro” y “motocicleta”; mientras que para las clases “persona”, “gato” y “perro”, el éxito fue menor. La precisión de predicción total de la red en las pruebas realizadas fue del 90%; sin embargo, la red tiene problemas en la predicción de las clases “gato” y “perro” principalmente.

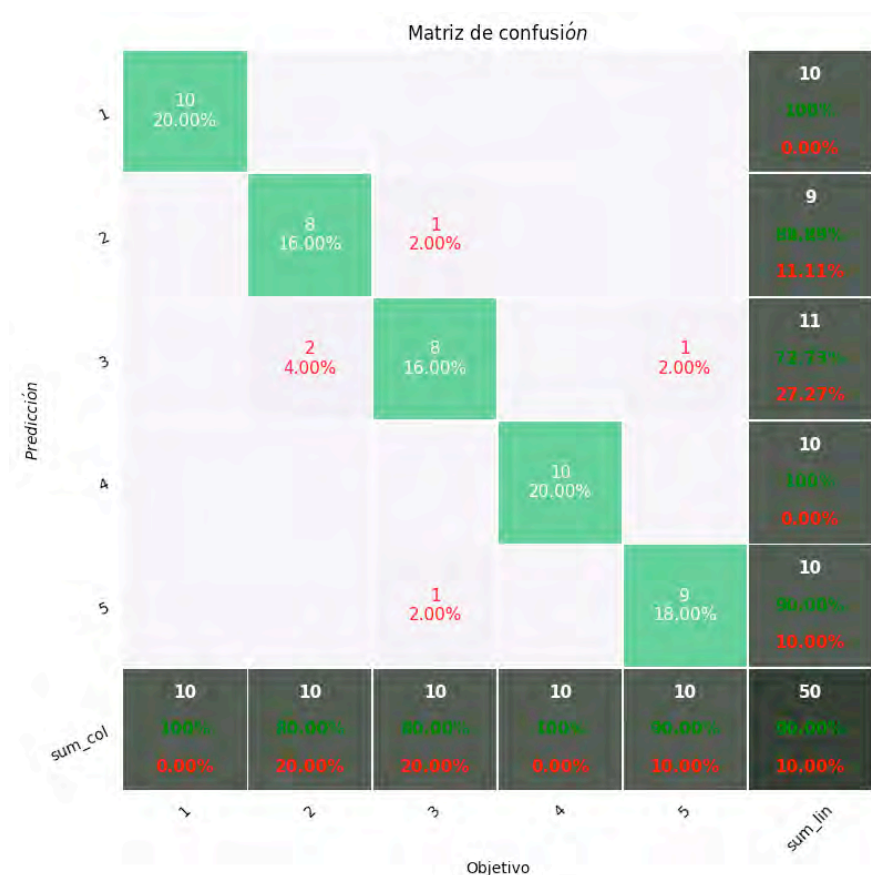


Figura 6 Matriz de confusión para la clasificación de imágenes utilizando el prototipo. Las etiquetas 1, 2, 3, 4 y 5, representan a las clases “carro”, “gato”, “perro”, “motocicleta” y “persona”, respectivamente.

La red tiene buen éxito de predicción cuando la imagen que capta la cámara contiene sólo una de las cinco clases, como se puede apreciar en las figuras 7 y 8. Sin embargo, la red puede tener problemas de predicción con imágenes que contienen más de una de las cinco clases.

#### 4. Discusión

La implementación de redes neuronales convolucionales en sistemas embebidos de bajo costo, para aplicaciones de visión artificial, es relativamente fácil con el uso del NCS. La instalación del software para ambos dispositivos es relativamente sencilla, sin embargo, el entrenamiento y compilación (archivo \*.graph) de la red son un poco complicados.





Figura 7 En la imagen se muestra una camioneta dentro del estacionamiento de la UAM-Azcapotzalco. El prototipo la clasifica dentro de la clase “carro” con una exactitud del 100%.



Figura 8 En la imagen se muestra a un alumno dentro de un laboratorio de la UAM-Azcapotzalco. El prototipo lo clasifica dentro de la clase “persona” con una exactitud del 100%.



La cantidad insuficiente de imágenes utilizadas para el entrenamiento de una red tan profunda como GoogleNet, produce un rendimiento pobre en la fase de prueba. Entrenar la red con un número grande de épocas, no implica obtener un mejor rendimiento en prueba, pero sí puede provocar que la red se adapte cada vez más a los ejemplos de entrenamiento causando el sobreajuste de la red. La CNN alcanza su máximo rendimiento en prueba en aproximadamente en 450 épocas y permanece constante hasta el final de entrenamiento, señal del sobreajuste que está presentando. Para garantizar un entrenamiento eficiente y un aumento considerable en el rendimiento en la fase de prueba, es necesario utilizar un conjunto de entrenamiento lo suficientemente grande; que se podría obtener aplicando la técnica de aumento de datos, la cual puede aumentar el tamaño del conjunto de entrenamiento 10 veces o más, haciendo a la red más robusta para evitar el sobreajuste.

En los conjuntos de entrenamiento y prueba existen imágenes que pertenecen a una clase pero incluyen elementos de otras clases, por ejemplo, en la figura 9 se muestra una imagen que pertenece a la clase “carro” que también incluye a las clases “persona”, “motocicleta” y “perro”. Imágenes como estas confunden mucho al clasificador y producen un rendimiento de prueba bajo. Eliminar este tipo de imágenes de los conjunto de datos, podría mejorar el rendimiento de la red; sin embargo, esto no sería una buena idea ya que este tipo de imágenes son muy comunes en calles y avenidas hoy en día. El clasificador desarrollado en el prototipo, solo considera la clasificación de una clase a la vez, no una combinación de ellas.

La reproducción en audio de la etiqueta del objeto clasificado, se realiza correctamente; sin embargo, se podría mejorar seleccionando diferentes tipos de voz y variando los parámetros de frecuencia y duración.

## **5. Conclusiones**

Los resultados obtenidos en este trabajo, indican que tanto la profundidad de la CNN como el tamaño del conjunto de entrenamiento, son cruciales para que la red

alcance un rendimiento alto en la fase de prueba; en otras palabras, para que la red tenga una buena capacidad de generalización.

El Neural Compute Stick, es un dispositivo muy confiable para la implementación de redes neuronales convolucionales profundas y su integración con sistemas embebidos de bajo costo, como la Raspberry Pi 3, es muy simple; sin embargo, hay que tener cuidado con la compilación y la descarga de la red.

El prototipo portátil desarrollado, tiene la capacidad de clasificar 5 clases de objetos diferentes y de reproducir una salida de audio describiendo la clase a la que pertenece el objeto. Sistemas con estas características, pueden ser de gran uso para desarrollar una gran cantidad de aplicaciones, por ejemplo, para ayudar a personas invidentes a encontrar objetos o para ayudar a niños de pre-escolar a aprender los nombres de los objetos, por citar algunos.

Como trabajo futuro se planea, evaluar el rendimiento del NCS con redes más profundas, como por ejemplo: ResNet 152. Redes entrenadas con conjuntos de datos que contienen miles o millones de imágenes a color de alta resolución, desarrolladas para la detección de objetos.



Figura 9 Imagen que contiene cuatro de las cinco clases utilizadas.

## 6. Bibliografía y Referencias

- [1] Caffe: <http://caffe.berkeleyvision.org/>, Agosto 2018.
- [2] Ciresan, D. C., Meier, U., Masci, J., Gambardella, L., Schmidhuber, J., Flexible high performance convolutional neural networks for image classification. In Proceeding of the Twenty-Second International Joint Conference on Artificial Intelligence, Vol. 2, 1237-1242, 2011.
- [3] eSpeak: <http://espeak.sourceforge.net/commands.html>, Abril 2018.
- [4] Everingham, M. and Van-Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results: <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>, Abril 2018
- [5] Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., Culurciello E., Hardware accelerated convolutional neural networks for synthetic vision systems. In Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), 257-260, 2010.
- [6] Girshick, R., Donahue, J., Darrell, T., Malik, J., Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 580-587, 2014.
- [7] He, K., Zhang, X., Ren, S., Sun, J., Deep residual learning for image recognition. In Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778, 2016.
- [8] Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications. CoRR, vol. abs/1704.04861, 2017.
- [9] Karpathy, A., Fei-Fei, L., Deep visual-semantic alignments for generating image descriptions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3128-3137, 2015.
- [10] Krizhevsky, A., Sutskever, I., Hilton, G., ImageNet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, 1097-1105, 2012.

- [11] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., Gradient-based learning applied to document recognition. In Proceedings of the IEEE, Vol. 86, Issue 11, 2278-2324, 1998.
- [12] Mollahosseini, A., Chan, D., Mahoor, M, H., Going deeper in facial expression recognition using deep neural networks. In Proceeding IEEE Conference on Applications of Computer Vision (WACV), 1-10, 2016.
- [13] Neural Compute Application Zoo (NC App Zoo): <https://github.com/movidius/ncappzoo/>, Abril 2018.
- [14] NC SDK, Intel® Movidius™: [https://movidius.github.io/ncsdk/tf\\_compile\\_guidance.html](https://movidius.github.io/ncsdk/tf_compile_guidance.html), Mayo 2018.
- [15] NCS Quick Star, Intel® Movidius™: <https://developer.movidius.com/start>, Marzo 2018.
- [16] Peemen, M., Setio, A., Mesman, B., Corporaal, H., Memory-centric accelerator design for convolutional neural networks. 31<sup>st</sup> International Conference on Computer Design, 13-19, 2013.
- [17] Prasoon, A., Petersen, K., Igel, C., Lauze, F., Dam, E., Nielsen, M., Deep feature learning for knee cartilage segmentation using a triplanar convolutional neural network. In International Conference on Medical Image Computing and Computer-Assisted Intervention, 246-253, 2013.
- [18] Raspberry Pi: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>, Marzo 2018.
- [19] Sankaradas, M., Jakkula, V., Cadambi, S., Chakradhar, S., Durdanovic, I., Cosatto, E., Graf, H., A massively parallel coprocessor for convolutional neural networks. 20<sup>th</sup> IEEE International Conference on Application-specific Systems, Architectures and Processors, 53-60, 2009.
- [20] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y., OverFeat: Integrated recognition, localization and detection using convolutional networks. Journal: arXiv preprint arXiv:1312.6229, 2014, Online, Available: <https://arxiv.org/pdf/1312.6229.pdf>.
- [21] Stanford Vision Lab, Stanford University, Imagenet. <http://imagenet.org>, Octubre 2017.

- [22] Strigl, D., Kofler, K., Podlipnig, S., Performance and scalability of gpu-based convolutional neural networks. In Proceeding 18<sup>th</sup> Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 317-324, 2010.
- [23] Suriyal S., Druzgalski C., Gautam K. Mobile assisted diabetic retinopathy detection using deep neural network. 2018 Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE/PAHCE), Marzo 2018.
- [24] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. Going Deeper with Convolutions. In Proceedings of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June, 2015.
- [25] Tafjira, N. B., Shun-Feng, S., Towards self-driving car using convolutional neural network and road lane detector. 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro--Mechanical System, and Information Technology (ICACOMIT), 65-69, 2017.
- [26] TensorFlow: <https://www.tensorflow.org/>, Agosto 2018.
- [27] Vasilache, N., Johnson, J., Mathieu, M., Chintala, S., Piantino, S., LeCun, Y., Fast convolutional net with fbfft: A gpu performance evaluation. arXiv preprint arXiv:1412.7580, 2014, Online, Available: <https://arxiv.org/pdf/1412.7580.pdf>.
- [28] Xu, X., Amaro, J., Caulfield, S., Forembski, A., Falcao, G., and Maloney, D. Convolutional Neural Network on Neural Compute Stick for Voxelized Point-clouds Classification. 10<sup>th</sup> International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2017.



# Apéndice D

## Constancias



**Figura D.1.** Constancia COMIA 2018.

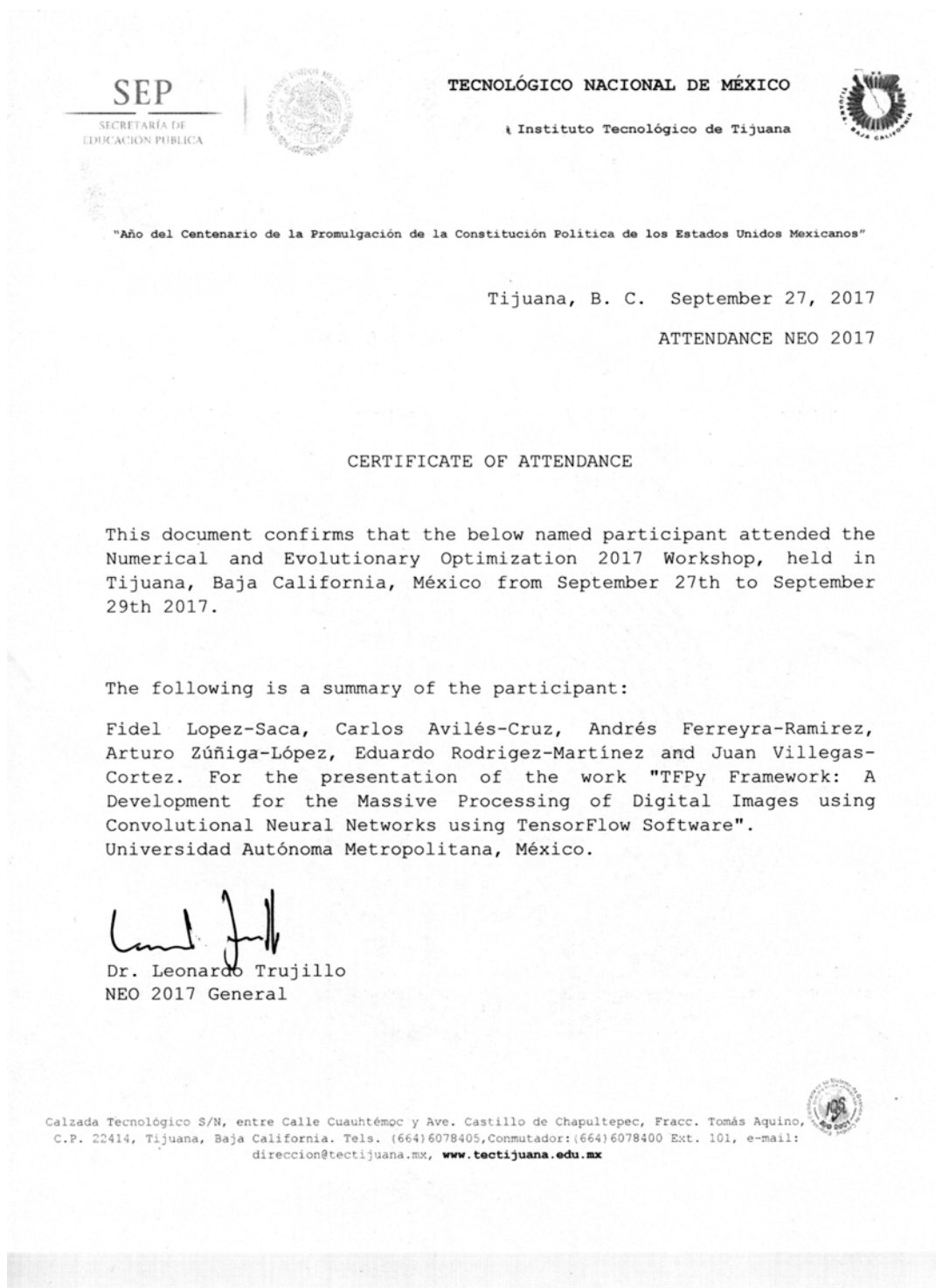


Figura D.2. Constancia NEO 2017.